SYSTEM AND METHOD FOR DELIVERING FINANCIAL SERVICES

PRIORITY APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/156,684 filed September 29, 1999 entitled "System and Method for Remotely Monitoring Hardware and Software Devices (NTDS Management Agents)" and incorporated herein by this reference.

This application is a continuation-in-part of co-pending U.S. Patent Application Serial No. 09/323,210 filed June 1, 1999, entitled "System and Method for Delivering Financial Services" which is a continuation of U.S. Patent Number 5,933,816, entitled "System and Method for Delivering Financial Services" issued on August 3, 1999.

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. Patent Application Serial No. 09/587,826 filed June 6, 2000 entitled "Methods and Systems for Automated Information Retrieval (AIRS)" and incorporated herein by this reference.

This application is related to US Application Serial No. 09/551,930, filed April 19, 2000, entitled "Platform-Independent Exceptions-Based Methods and Systems for Remotely Managing Nodes Within a Communications Network," and is incorporated herein by reference.

This application is related to ~~Attorney Docket No. CITI0209/196411, filed simultaneously~~, entitled "System and Method for Delivering Financial Services," and is incorporated herein by reference.

NOTICE OF COPYRIGHTED MATERIAL IN DISCLOSURE

A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or of the patent disclosure, as it appears in the Patent and

Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The invention generally relates to a system and method for delivering financial services and, more particularly, to a system and method for delivering financial services to a plurality of different devices.

BACKGROUND OF THE INVENTION

Banks and other institutions that provide financial services are facing increased amounts of competition and are being pressured to provide a greater diversity of services to their customers. Not too long ago when customers traveled to the bank to make all of their transactions, the bank could focus on the customer-bank teller interaction to improve the quality of services. The bank could improve the quality of service by staffing a larger number of tellers at peak times and by offering drive through services. Banks developed internal computer systems and provided their tellers with staff terminals so that the bank tellers could access the books of the bank when they were entering customer transactions.

This simplified model of banking, while still in existence, has been greatly expanded. In addition to the bank tellers, banks provide Automated Teller Machines (ATMs) so that customers can perform transactions at literally any hour of the day. The locations of the ATMs are not limited simply at the bank's branch locations but can be found, for instance, in shopping malls, airports, grocery stores, hotels, and building lobbies. Since the ATMs must access the books of the banks to allow customers to perform their transactions, banks had to provide an interface between the ATMs and the bank's internal computer system that allows the ATMs limited but secure access to the bank's books.

The model for providing financial services has been expanded even further to enable home banking. With home banking, a customer can access his or her personal account and perform transactions, such as bill paying, money transfers, etc., in the convenience of one's home through the customer's personal computer. To enable home banking, banks had to provide an interface between the bank's internal computer system and the customer's

personal computers to allow limited and secure access to the bank's books. Due to the differences between ATMs and personal computers, the interface for the personal computers is typically separate and distinct from the interface provided for the ATMs.

One difficulty facing banks is that they must provide a first interface between the bank's internal computer system and the staff terminals, a second interface between the internal computer system and the ATMs, and a third interface between the internal computer system and the personal computers. Each of these interfaces or platforms adds complexity to the bank's overall computer system and each competes with the other for access to the bank's books. This added complexity is significant since the amount of resources that the bank must devote toward maintaining its computer system is increased due to these three separate platforms.

The complexity of a bank's computer system will only grow as banks continue to provide more and more services. One area of service that banks are already beginning to explore is the granting of access to the bank's books by devices other than personal computers, such as screen phones or personal data assistants (PDAs). Another area of service that banks are contemplating is the granting of access to the bank's books through the Internet, such as through an Internet service provider (ISP) or other external service provider (ESP). These additional remote devices and the connection to the ISP would further complicate the bank's internal computer system and would require the bank to devote more resources in maintaining and upgrading its computer system.

In addition to additional channels of access to the bank's books, banks have been expanding the types of services that can be accessed by a remote device. In addition to traditional checking and savings accounts transactions, banks are also enabling the paying of bills, the buying and selling stocks, the quoting of stocks, as well as other types of services through the ATMs, personal computers, or other remote devices. Each expansion into another type of service requires a significant amount of modification to the software and possibly hardware in the bank's internal computing system. These additional services, although necessary to remain competitive, require a considerable amount of work on the bank's computer system.

The complexity in offering financial services will also be compounded as more and more services are being provided in an international market. As individual national markets in the world continue to merge together, both businesses and individual customers will have an increased need and desire to access their account information from another country. The platform or platforms needed to interface with the banking systems of other countries will further tax the ability of a bank to maintain its computer system and its books.

Another difficulty facing a bank entering another country is that the bank must, in effect, create a new computer system for each country that it enters. Each country has its own unique regulatory and legal environment which dictates the manner in which financial services must be performed. The bank cannot simply duplicate a computer system operating in another country but rather must specially tailor each computer system to the regulatory and legal environment of that country. This extra amount of effort required to shape a computer system according to the rules and laws of its home country consumes more of the bank's valuable resources.

The creation of computer systems for banks in other countries is complicated by the difference in languages. Because of a language difference, the interface between the computer system and the customer, such as through a graphical user interface (GUI), will vary according to the national language or languages of a particular country. These differences in how customers interface with the bank's computing system is not limited simply to words and different alphabets but also encompass manifestations of language due to differences in culture or norms of a particular country. These manifestations of language, for instance, may dictate the selection of certain colors in a GUI, the use of a particular set of symbols, and the selection of certain audio indicators, such as beeps or other tones.

In view of a desire to allow international access to the bank's books, each computer system should also be able to receive and send communications from the other computer systems, which possibly may be in numerous languages. For instance, the bank's books which are stored in the United States may need to be accessed by a newly installed banking system in Thailand. Before this access is possible, however, the banking system in the United States may need to be modified to recognize the bank's computer system in Thailand.

Consequently, with the introduction of each computer system in one country, the computer systems in all of the other countries may need to be modified accordingly.

As discussed above, banks are increasingly providing new types of access to the bank's books and are providing new services in an ever-increasing number of countries. All of these changes in services and access to services are accomplished by rewriting the application which governs all operations of the bank's internal computer system. Due to the extremely sensitive nature of the data in the bank's books and the need for total accuracy, any modification to this application must undergo an inordinate amount of testing. Since the application covers all aspects of the system, any modification to one part of the application which may cover only a minor aspect of the system can potentially have an effect on any other part of the system. The testing required for any modification, even for just a minor upgrade, must therefore be performed on the entire system.

Thus, a need exists for a computer system or method that has a reduced amount of complexity yet offers access to various remote devices and enables the expansion of access to new types of devices. A need also exists for a computer system or method that can offer new or modified services more easily with less testing. A need also exists for a computer system or method that can more easily accommodate the legal and regulatory environment of a host country and which can more easily interconnect and communicate with the systems in other countries.

Further, there is a need to get information from various types of devices, such as ATMs, to a central server, so that the automated teller system or systems can be monitored remotely. For example, the bank may have several thousand ATMs, which are hardware devices with software running in them, deployed in the field and which include many potential points of failure. The bank needs to be able to monitor and manage those devices from a central location. There is a great deal of information residing on those systems, and the bank needs to be able to get at the information and know when something goes wrong.

SUMMARY OF THE INVENTION

The invention, in a preferred embodiment, is a system and method for delivering financial services to a remote device. Through the remote device, a customer or employee of

a financial institution can select a mini-app dialog component to perform a function. Preferably, each function that may be performed is represented by a separate mini-app dialog component. Upon selection of a function, the mini-app dialog component collects information needed to perform the requested function and instantiates a transaction executor component to carry out the function. The remote device may comprise any type of device, such as a personal computer, screen phone, ATM, personal data assistant, or an internal staff terminal. The remote device may access the system in a variety of ways, such as through an external service provider, through the Internet, or through dial-up access. Thus, the system provides a single base for interfacing with all types of remote devices.

In generating graphic interfaces, the system and method preferably separate content from format to accommodate variations in the remote devices. The system includes a presentation manager which maps messages from a canonical representation into the format desired for a particular remote device. The content of the messages is regulated through a language man component. In response to a request for a named phrase, the language man component provides the phrase in the language and the content specific for that customer and that remote device. As a result, the system and method can provide state of the art user interfaces, can provide interfaces consistent for a financial institution, and can allow a customer to custom design a user interface.

The system and method operate in sessions with a session bubble instantiated for each session with a remote device. After receiving an initial contact with a remote device, a session controller instantiates a session component to manage resources for the session bubble. The session component, in turn, instantiates a number of components for the session, such as a wome mat component, front door man component, rule broker component, and acquirier component. The wome mat component sends a logon message to the remote device and instantiates a profile transaction executor component to authenticate a customer. A navigation shell component notifies the remote device of the list of available functions, such as cash withdrawal or bill payment, and instantiates a mini-app dialog component based on the function selected through the remote device. To coordinate communications with the plural sessions that may occur simultaneously, a touch point interface component routes incoming messages from remote devices to the appropriate session bubbles and a back door

man component coordinates messaging between the various sessions and an external service provider.

The system and method employ a rule broker component that other components within the system may query to obtain an answer to any question that might arise. The rule broker component registers rule authorities as the answerers of questions and directs the querying component to the rule authority for the answer. The rules within the system and method may be modified independently from other application components. With the rule broker, regulatory or business rules can be easily added, changed, or modified with only a minimal impact on overall operations.

The system and method are operable in conjunction with any legacy application that may already exist. A legacy app bridge component converts data from a canonical representation to the global data structure needed by the legacy applications. On an exit from a legacy application, the legacy app bridge component translates the data from the legacy application into the canonical representation. The legacy app bridge component therefore enables the continued operation of the legacy applications.

In a system monitoring and management aspect of an embodiment of the present invention, use is made of an agent set that provides a communication mechanism such that the bank can talk to its ATMs and can query them for their status. In addition, an embodiment of the present invention makes use of the concept of instrumentation in which software that is essentially resident on the ATMs monitors the hardware devices that are part of the ATM. When those hardware devices report a problem, the software is alerted. An embodiment of the present invention includes the concept of instruments which are addressable entities that can be queried about the status of any particular item on the ATM.

The system management services for an embodiment of the present invention include, for example, a management protocol agent, a command dispatch agent, and a status monitor agent. An external system management component, sends a management request relative to a managed component via an external interface to a management protocol agent. The management protocol agent receives and translates the request from a remote system management protocol into a specific command for the managed component, such as an

inquiry command, a stop command, or a start command, for the command dispatch agent and sends the translated command to the command dispatch agent.

The command dispatch agent for an embodiment of the present invention executes the command, for example, by obtaining the managed component from a component registry, with which the managed component was previously registered, and dispatching the command to the managed component. Executing the command by the command dispatch agent involves, for example, collecting one or more instruments owned by the managed component, inquiring about a value of one or more instruments owned by the managed component, and/or obtaining a status of one or more instruments owned by the managed component.

In an embodiment of the present invention, upon execution of the command, the external system management component is provided, via an external interface, with a response to the management request, such as the status of one or more instruments owned by the managed component or an acknowledgment of a stop or start command for the managed component. The response is provided to the management protocol agent, which translates the response into the remote management system protocol format and delivers the formatted response to the external system management component.

The status monitor agent for an embodiment of the present invention monitors one or more managed components in regard to instrument variables of the managed component and/or events relative to the managed component. With regard to instrument variables, the status monitor agent receives notification of a change in value of one or more instruments owned by the managed component, for example, by registering for notification of the change. Such instruments include, for example, counter instruments, bounded counter instruments, status instruments, and control instruments. With regard to events, the status monitor receives notification of events relative to the managed component by registering with an event broker for notification of the events. In addition, the status monitor agent can periodically poll the managed component to determine if a local action is required.

In an embodiment of the present invention, an alarm is generated relative to the managed component, for example, by generating an event by an instrument owned by the managed component through the managed component or by the event broker. In addition, a local action or inquiry can be initiated through the command dispatch agent by a rule based

state machine maintained by the status monitoring agent. Generating the alarm can also involve, for example, sending an unsolicited notification of an event or change in an instrument via an event notification interface published by the status monitoring agent. The alarm is translated by the management protocol agent into a remote system management protocol format for the external system management component, and the formatted alarm is provided to the external system management component via an external interface.

It is therefore an object of the present invention to provide a delivery system and method that provide a common application base for all remote devices.

It is also an object of the present invention to provide a delivery system and method that provide a set of re-usable components that can be easily modified or expanded to fit the needs of a particular environment.

It is another object of the present invention to provide a delivery system and method that provide and can easily maintain state of the art user interfaces.

It is a further object of the present invention to provide a delivery system and method that reduce development and maintenance cycle time.

It is yet another object of the present invention to provide a delivery system and method that are capable of supporting existing remote devices having legacy applications.

It is an additional feature and advantage of the present invention to provide a method and system for remotely monitoring and managing hardware and software devices, such as ATMs and home banking servers, from a central location.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram of a delivery system according to a preferred embodiment of the invention connected to a plurality of remote devices.

Fig. 2 is an overall block diagram of the delivery system of Fig. 1.

Figs. 3A to 3C are flow charts depicting operations of the delivery system in starting a banking session.

Figs. 4A to 4C are partial block diagrams of the delivery system depicting the operations shown in Figs. 3A to 3C.

Figs. 5A to 5D are flow charts depicting operations of the delivery system in authenticating a customer.

Figs. 6A to 6C are partial block diagrams of the delivery system depicting the operations shown in Figs. 5A to 5C.

Figs. 7A and 7B are flow charts depicting operations of the delivery system in the selection of a mini-application by a customer.

Figs. 8A and 8B are partial block diagrams of the delivery system depicting the operations shown in Figs. 7A and 7B.

Fig. 9 is a block diagram of a NetCAT delivery system according to a preferred embodiment of the invention for use in providing CAT software to foreign CATs.

Fig. 10 is a block diagram of a CAT delivery system according to a preferred embodiment of the invention.

Fig. 11 is an exemplary diagram of an interaction between a legacy app bridge component 84 in the system of Fig. 2 with a legacy application.

Fig. 12 shows a component object model which illustrates an example of key components and the relationship between the components for the remote system management aspect for an embodiment of the present invention.

Fig. 13 shows a use case scenario which illustrates an example of system manager command processing for an embodiment of the present invention.

Fig. 14 shows a use case scenario which illustrates another example of system manager command processing for an embodiment of the present invention.

Fig. 15 shows a use case scenario which illustrates an example of status of cash status event processing for an embodiment of the present invention.

Fig. 16 shows a component object model which illustrates an example of a component factory and its collaborating components and the relationships between them for an embodiment of the present invention.

Fig.17 shows a component object model which illustrates an example of instruments for an embodiment of the present invention.

Fig. 18 shows a component object model which illustrates an example of a managed component for an embodiment of the present invention.

Fig. 19 shows a component object model which illustrates an example of a monitored component for an embodiment of the present invention.

Fig. 20 shows a use case scenario which illustrates an example of the component factory startup for an embodiment of the present invention.

Fig. 21 shows a use case scenario which illustrates an example of create component process for an embodiment of the present invention.

Fig. 22 shows a use case scenario which illustrates an example of an instrument rendezvous for an embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Reference will now be made in detail to the preferred embodiment of the invention, an example of which is illustrated in the accompanying drawings. The invention is described with reference to a system 10 for use by a bank, although the system 10 may be employed by any type of institution offering financial services. The financial system 10 includes a delivery system 12 for providing financial services to a variety of remote devices. These remote devices include a screen phone 14, an automated teller machine (ATM) 16, such as Citibank's CAT/CASST terminals, a personal computer 18, or a personal data assistant (PDA) 20. The remote devices can practically be any type of device and can be installed with any suitable software for communicating with the delivery system 12, such as a standard web browser or any other third party software product. The remote devices that the delivery system 12 can provide financial services to is therefore not limited to any particular class or type of remote device but instead may include any future device or system. Further, the delivery system 12 provides services not only to customers of a financial institution but may also provide services internally to the institution, such as at staff terminals 26.

The delivery system 12, furthermore, provides financial services over a plurality of different delivery networks. As an example, the delivery system 12 may deliver financial services to the screen phone 14, personal computer 18, or PDA 20 via dial-up access or through an application server, such as the home services delivery system (HSDS), which is disclosed in U.S. Patent No. 5,485,370 to Moss et al. and which is hereby incorporated by reference. Alternatively, the delivery system 12 may provide financial services to remote devices 24 through an Internet Service Provider (ISP) 22 or an on-line service provider 22, such as through the Internet or World Wide Web. The delivery system 12 advantageously is able to provide financial services over a variety of communication paths, such as the Internet, a land-line telephone network, a cellular network, or a cable network, and can be easily modified to operate over new transmission paths or new transmission protocols.

I. Service Sets And Components

With reference to Fig. 2, a delivery system 12 according to a preferred embodiment of the invention comprises plural sets of service components. These sets of service components

include a touch point and display set 30, a touch point interface services set 40, and a touch point services set 50. In general, the touch point and display set 30 provides actual customer display and input facility and the touch point interface services set 40 provides an interface to the touch point services set 50. The touch point services set 50 provides presentation mapping and front door security for the delivery system 12. The delivery system 12 also includes a peripheral device services set 60 providing peripheral device interface and management services. A system services set 70 provides logging, event brokering, service registry and crypto services and a dialog services set 80 provides woming, navigation shell and application specific dialogs. A transaction services set 90 provides transaction coordination and ESP message formatting and an external service provider interface services set 100 provides message sequencing and ESP interface protocols. A customer services set 110 provides customer identification, relationship, account and issuer services and a business services set 120 provides rule brokering, language, and acquirer services. A session services set 130 provides session start up and session and delivery vehicle context.

A. Touch Point and Display Set 30

The touch point and display set 30 provides the actual customer display and input facility on the remote device. The touch point and display set 30 includes a touch point and display component 31 that displays pages on the remote device screen and sends customer inputs to the delivery system 12. The touch point and display component 31 is responsible for managing the link/session level protocols with an application server on the remote device. The touch point and display component 31 also decodes the server interface protocol and outputs a page to the local screen of the remote device. The touch point and display component 31 acquires customer inputs, including choice selections and forms input, encodes the input in the server interface protocol, and sends the customer input to the touch point interface set 40. For Internet sessions with the delivery system 12, the touch point and display component 31 preferably comprises a web browser that handles protocols such as TCP/IP, HTTPS, and, less preferably, FTP.

B. Touch Point Interface Services Set 40

The touch point interface services set 40 provides an interface to the touch point services set 50 and includes a touch point interface component 41. The touch point interface component 41 is responsible for managing the link/session level protocols with a remote device. The touch point interface component 41, for instance, notifies the session services set 130 to start a new session on initial contact from a remote device. The touch point interface component 41 also encodes messages in the interface protocol, sends messages to the touch point services set 50, and decodes messages received from the touch point services set 50. The touch point interface component 41 further routes received messages to an appropriate session front door man component 51 of the touch point services set 50. For Internet sessions with the delivery system 12, the touch point interface component 41 preferably comprises a web server which handles the protocols such as TCP/IP, HTTPS, and, less preferably, FTP.

C. Touch Point Services Set 50

The touch point services set 50 is responsible for final device specific presentation layout and front door security and includes the components of the front door man component 51 and a presentation manager component 52. The front door man component 51 guards access of a remote device into a session. For remote sessions, the front door man component 51 adds a session security token to outgoing messages and verifies the session security token for incoming messages. For sessions with a CAT/CASST 16, the front man component 51 may simply pass through communications. Although the front door man component 51 has been shown as a single component, the front door man component 51 preferably comprises a separate component for each type of remote device.

The presentation manager component 52 is responsible for mapping a canonical representation of information on pages into a specific style layout in a device specific presentation format. Thus, the same application can have different presentation styles on different device types. For instance, the same application may have different presentation styles depending on whether the application is displayed on a personal computer 18, a PDA 20, a screen phone 14, a CAT 16, a third party kiosk terminal, or another type of remote device. The style templates can be customized by region to support local cultural differences in areas such as color schemes, graphics, icons, and font sizes. The presentation manager

component 52 maps tagged phrases and data from the application into specific fields of a particular page template referenced by the application. A template controls the layout and representation of frames on a page, multi-media elements, choice and data fields, and input forms on the page for a specific style and device type. The presentation manager component 52 also encodes the resulting page in the device specific format for the particular remote device and sends the page to the front door man component 51. The presentation manager component 52 also receives incoming messages from the remote device, converts choice information and form fields from the device specific format to a tagged canonical representation, and routes the representation to the appropriate component within the dialog services set 80. The presentation manager component 52 uses delivery system specific templates to enforce consistent layout styles across pages having similar choices, data fields, and forms. A template can be the superset of all possible objects on a page since the presentation manager component 52 can "drop out" fields and choices which are not associated with any data. Reference is made to a related application serial number 08/741,121, "Method and System for Automatically Harmonizing Access to a Software Application Program Via Different Access Devices," filed October 30, 1996 which is incorporated herein by reference.

D. Peripheral Device Services Set 60

The peripheral device services set 60 is responsible for handling application requests for peripheral device services and for managing the software components that handle such requests. Peripheral device services set 60, for instance, provides a custom high-level application interface to peripheral devices associated with a remote device, such as the CAT/CASST 16. The peripheral device services set 60 includes a peripheral device handler component 61, a peripheral device manager component 62, and a session device manager component 63.

The peripheral device handler component 61 represents and controls a specific kind of connected peripheral hardware device. Several kinds of peripheral devices may be connected to a service delivery platform for the CAT/CASST 16. The peripheral device handler component 61 preferably comprises a plurality of peripheral device handler components 61

with each peripheral device handler component 61 providing a generic device management interface and a specific service interface. Further, the peripheral device handler component 61 is not limited to a single component, but may comprise specific subcomponents to interface with its associated peripheral hardware device. The peripheral device handler component 61 loads and activates needed subcomponents and initializes the specific peripheral device hardware. The peripheral device handler component 61 maintains persistent peripheral-specific management statistics and reports these statistics as well as status of the peripheral device upon request. The peripheral device handler component 61 also notifies interested parties of changes in peripheral device status and recovers peripheral device hardware functionality after a failure. The peripheral device handler component 61 finalizes by releasing any needed system resources and deactivates and unloads subcomponents. In addition to its device management responsibilities, the peripheral device handler 61 also provides application services. For instance, the peripheral device handler component 61 tests the connected hardware device for correct operation, normalizes the service interface so that similar devices share a common interface, and translates application requests into detailed hardware device requests. As will be appreciated to those skilled in the art, the specific nature of the services rendered by the peripheral device handler component 61 depends upon the specific hardware device type.

The peripheral device manager component 62 manages the components that interface with the connected peripheral devices. The peripheral device manger component 62 loads the peripheral device handler component 61 for the connected devices during startup and initializes the peripheral device handler component 61 during startup. The peripheral device manager 62 notifies interested parties of changes in peripheral device availability, finalizes the peripheral device handler component 61 during shut down, and unloads the peripheral device handler component 61 during shut down. In addition to its responsibilities for device management, the peripheral device manager component 62 also provides application services. For instance, the peripheral device manager component 62 coordinates usage of the peripheral devices by customers versus diagnostics and serializes application requests to each peripheral device. The peripheral device manager component 62 also routes each application request to the appropriate peripheral device handler component 61 and reports status of all connected

peripheral devices upon request.

The session device manager component 63 is an in-session component that coordinates the control access to control devices via an acquisition mechanism. Upon request, the session device manager component 63 first determines the availability and capability of the acquired device and returns the device reference to the client. The session device manager component 63 queries the peripheral device manager component 62 to determine devices available to the system, queries the delivery capabilities to determine the available remote devices and creates instances of those devices for use by session components and services acquired device requests from the dialog services set 80 for requested type of interface for a specific device. The types of interfaces supported, for instance, include management interface, application interface, and diagnostic interface.

The delivery system 12 is not limited to any particular type of peripheral device. Further the delivery system 12 is not limited to peripheral devices that are associated with any particular type of remote device, such as the CAT/CASST 16, but rather may be associated with any remote device. The peripheral devices include, as examples, a touch screen, screen display, form printer, card reader, PIN encrypter, envelope depository, cash dispenser, speech generator, and sound generator. The peripheral devices also may include an audio generator, video player, proximity detector, and a biometric scanner. As will be apparent to those skilled in the art, the status of a peripheral device and statistics associated with that device will vary with the particular peripheral device. For instance, with a card reader, the status may be up/down and capture bin full. The statistics associated with a card reader may include the number of cards read, the bad reads, cards captured, long time-outs and short time-outs. As another example, the status of a depositor may include up/down, ink low, or bin full and the statistics for the depositor may include the number of envelopes captured.

E. System Services Set 70

The system services set 70 provides common services for all sessions within a server, including logging, event brokering, service registration, and cryptographic services. The system services set 70 includes a process controller component 71, a logger component 72, an event broker component 73, a services registry component 74, a crypto man component 75, an

instrumentation component 76, a system management agents component 77, and a test manager component 78.

The process controller component 71 starts up all the non-session system service and peripheral device management processes in the delivery system 12. The components that the process controller starts includes the logger component 72, the event broker component 73, the services registry component 74, the crypto man component 75, the instrumentation component 76, the system management agents component 77, and the test manager component 78. The process controller component 71 further starts up the peripheral device manager component 62 in the peripheral device services set 60 and a session controller component 131 in the session services set 130.

The logger component 72 writes and manages log files and works in conjunction with an NT log facility. The logger component 72, for instance, adds standard headers to log entries and writes the log entries to a log.

The event broker component 73 provides a way for a business to do specialized processing of events for the purpose of monitoring and acting upon activities in a server. Local business provided components can register with the event broker component 73 to receive specified events. The event broker component 73 evaluates filtering rules associated with events and then calls the registered component as a result of a rule succeeding. The event broker component 73, for example, could decide when to send notifications to a system management system.

The services registry component 74 registers the mini-apps and legacy app bridges that are available. The services registry component 74 has a CreateComponent function that, given a well-known name for a service, will look up the full class name and create the component. The services registry component 74 works in the context of the procedures for software distribution and cutover/fallback of releases in order to maintain a registry of the mini-apps and legacy apps that are currently available. The services registry component 74 also provides information to a navigation shell component 82 within dialog services set 80 about the mini-apps and legacy apps that are currently available.

The crypto man component 75 performs cryptographic functions necessary to handle security. The crypto man component 75 manages secret keys associated with external service

providers and performs authentication of public key certificates. The crypto man component 75 holds security keys for each external service provider, which may be multi-level keys for each external service provider. Further, the keys may be shared secret or private key associated with a public key. The crypto man component 75 also updates keys and uses keys to generate message MAC and encrypt message. The crypto man component 75 also encrypts and re-encrypts customer PIN/TPIN.

Many components of the delivery system 12 need to update counters or provide some means by which they may be monitored or controlled, especially components that need to support being monitored and controlled by the system management facilities. Several instruments allow interested components to observe changes in other components. Each instrument provides a point of contact or rendezvous between an instrument updater and its interested observers. Whenever an instrument updater changes the instrument value, the interested observers are notified of the change, giving the opportunity to observe the changed instrument value. All instruments are created and maintained by the instrument manager component 76. Both instrument updaters and instrument observers obtain references to instruments from the instrument manager component 76. Each kind of instrument has a publisher that defines the name of the instrument and the value of the various instrument properties. The instrumentation component 76 performs an important function of keeping a record of counters and controlled variables in a persistent store. The supported instruments include, but are not limited to, the counter instrument, bounded counter instrument 348, status instrument, and control instrument. The instrument component 76 creates and maintains counters, maintains a value, publishes a list of the status values and names, registers and unregisters value observers, and increments or decrements a value. The instrumentation component 76 also notifies registered observers when a value changes and notifies registered observers when a limited counter value crosses a threshold, such as a lower bound or an upper bound threshold.

The system management agents component 77 comprises three agent components: a management protocol agent, a command dispatch agent, and a status management agent. The management protocol agent interfaces with an external system management product on the system 10 and translates a specific system management protocol to or from a format

supported by the command dispatch agent and the status monitor agent. The management protocol agent translates an incoming management request into an inquiry or modify for the command dispatcher agent, translates a system management alarm from the status monitoring agent into the remote system management protocol, and supports secure access to the management server. The command dispatch agent translates requests for actions or status into the proper control instrument variables needed to control a component or retrieve its status. The command dispatch agent translates inquiries/modify requests to proper instrumentation component instrumentation objects, such as control variable, counter, and status indicator. The status monitor agent monitors status instrument variables and events, determines if an external system management product needs to be notified, and sends any important "alarms" to the external system management product. The status monitoring agent registers for events from the event broker component 73, registers for changes to instruments, state machine correlates and filters information, uses instruments for some local action or inquiry, and sends a "higher level alarm" to the management protocol agent and/or to an event broker management protocol agent.

The test manager component 78 manages the testing and tracing of components in the system 12. The test manager component 78 collects information from the various components in the system 12 by wiring its into them during component creation. Then, the components that have been wired for test report method entries and exits to the test manager component 78 during their operation. The configuration of which components are under test or trace can be driven by scripts or by an on-line test management user interface. The test manager component 78 records information reported by the components under test in a log or it can report the test results to the tester through the test management user interface. The test manager component 78 therefore knows which components are under trace and test and wires new components for tracing and testing.

F. Dialog Services Set 80

The dialog services set 80 is responsible for the semantic content and interaction with the customer and for initiating transactions on the customer's behalf. The dialog services set 80 includes a wome mat component 81, at least one navigation shell component 82, at least

one mini-app dialog component 83, and at least one legacy app bridge component 84. Although the navigation shell component 82, the mini-app dialog component 83 and the legacy app bridge component 84 have been shown as single components, each of these components may comprise a plurality of components.

The wome mat component 81 outputs the initial wome page to the customer and collects customer identity and preference information. After determining the issuer of the customer ID and possibly authenticating the customer, the wome mat component 81 instantiates several customer services objects to hold information about the customer and then starts a navigation shell component 82 which carries out the next level of dialog with the customer. The wome mat component 81 establishes connection sessions with a back door man component 101 in the ESP interface services set 100 as needed by a session. The wome mat component 81 also acquires devices needed by the session and creates a scam transaction executor to handle unsolicited scam events from a host. The wome mat component 81 presents an out of service or wome page, enables a card reader, and waits for card read events. If the card event is an administration card, the wome mat component 81 instantiates an administrative wome mat component. The wome mat component 81 collects various information from the customer including language choice and other preferences, such as navigation style. The wome mat component 81 also collects customer ID information, such as CIN/PIN and public key certificate, in a manner consistent with the customer remote device and mode of access, such as dial-in or Internet. The wome mat component 81 handles retries if errors occur on customer identity input, for instance by re-reading a card, and asks customer ID component 111 for issuer. The wome mat component 81 instantiates a profile transaction executor component 91 to authenticate the customer and get the customer's relationships or customer profile. This process typically involves interactions with the issuer external service provider, but may alternatively be performed locally based on information in a SmartCard. The transaction executor component 91 instantiated by the wome mat component 81 will instantiate the following customer service components: customer ID component 111, customer relationship component 113, account component 115, and issuer component 112. The wome mat component 81 will also instantiate a transaction record queue component 91, initialize legacy app bridge components 84, and start a navigation shell

component 82 based on delivery capabilities, acquirer rules, and customer preferences.

The wome mat component 81 may rely on separate micro-app dialog subcomponents to do some parts of the dialog that may be common to several business functions or which may vary depending on the remote device peripherals. For instance, the wome mat component 81 may rely on a hello screen micro app, a language select micro app, and a get PIN customer data micro app.

The wome mat component 81 may do four things for customer authentication based on acquirer rules and the type of customer ID, such as public key certificate, ATM card, credit card, on-us, or off-us. The wome mat component 81 may provide immediate local authentication using public key certificates or may provide immediate authentication with the issuer, waiting for a response. The wome mat component 81 may also provide background authentication with the issuer while going on to the navigation shell component 82 or may defer authentication to the first transaction. With deferred authentication, the wome mat component 81 may need to instantiate a default customer relationship component 113 and a default set of product types, such as checking, savings, or credit card. If a rule broker component 121 does not have a registered issuer for the card/CIN prefix number, a customer ID component 111 is instantiated and marked invalid, further authentication of the customer is skipped, and a navigation shell component 82 for invalid customers is started. Invalid customers may still be allowed to use certain information only in mini-app dialogs.

The navigation shell component 82 informs the customer of the range of mini-apps that are available and provides top level navigation across these applications. The navigation shell component 82 assigns a frame space within which a mini-app runs. To support complex grouping of functions or a variety of navigation styles, the navigation shell component 82 may contain shells within shells. The navigation shell components 82 available for selection by a customer include linear, which guides customers through detailed question and answer steps; nonlinear broad branching, such as pull-down menus; preferred, such as customer specified short cuts; or query, which may include a search engine or natural language searching capabilities. The navigation shell component 82 obtains lists of possible services available from services registry component 74, checks rules to see what services are actually available in the current system context, and makes the customer aware of the range of mini-

apps available. The range of mini-apps available will be based on the customer's relationship, the issuer/acquirer rules, and the set of dynamically registered mini-apps. The mini-apps may be organized and identified by the navigation shell component 82 with names, icons, or any other type of representation. The navigation shell component 82 instantiates additional navigation shell components 82 as necessary and instantiates mini-app dialog component 83 as requested by the customer. The navigation shell component 82 supports switching between concurrently active mini-app dialogs and, at the end of a session, instantiates and calls end of session mini-app. The delivery system 12 preferably supports the customer leaving a mini-app to enter the navigation shell component 82 and to start another mini-app, while leaving the former mini-app suspended in its current context state. The customer can later exit from the new mini-app and go back to the former mini-app or can switch between multiple concurrently active mini-apps. In an environment where the screen has imbedded frames, a main navigation shell component 82 may, for example, invoke one or more sub shell components 82 to control individual frames.

The mini-app dialog component 83 manages the dialog with a customer for a specific business function in a specific dialog style. The mini-app dialog component 83, for instance, may manage the business functions of transferring funds or bill payment in the styles of question and answer or forms. The mini-app dialog component 83 presents information and choices to the customer and collects and validates customer inputs. The mini-app dialog component 83 is responsible for the content of information on pages and the flow of the customer interaction, but preferably not the style and layout of the presentation. The mini-app dialog component 83 may comprise several different mini-app dialog components 83 with different dialog styles for the same business function. The mini-app dialog components 83 may support different modes of the customer entering information, such as guiding the customer through detailed question and answer steps or forms with multiple input fields. After collecting the necessary customer inputs for a particular business function, the mini-app dialog component 83 uses a transaction executor component 91 to carry out the function by doing transactions with external service providers and operating peripheral devices, such as a cash dispenser or depositor. The mini-app dialog component 83 implements the customer-visible control flow for a particular function in a specific dialog style. The flow may be

tailored based on the customer relationship and on various countries/business rules. The mini-app dialog component 83 uses a language man component 122 within the business services set 120 to do translation of phrases into target languages for display or print. The mini-app dialog component 83 assembles phrases and formatted data into pages, for display or print, with each page constructed in a canonical format by setting properties of named objects within named templates. The mini-app dialog component 83 sends pages to the presentation manager component 52 which handles the final style and layout for the specific remote device. The mini-app dialog component 83 collects customer inputs and validates customer inputs using business rules. Validation, for instance, includes basic field validations as well as cross-field validations. The mini-app dialog component 83 instantiates and calls transaction executor components 91 to do transactions with external service providers and also operates remote devices, such as a cash dispenser or a depositor, needed by the business function. The mini-app dialog component 83 queues transaction data for printed record and increments transaction counters in the instrumentation component 76. A mini-app dialog component 83 may, for instance, use separate mini-app dialog subcomponents 83 to do some parts of the dialog that may be common to several business functions, such as PIN entry, account resolution, and entering currency amount.

The legacy app bridge component 84 is a bridge that enables a legacy application set to operate in the delivery system 12. The legacy app bridge component 84 translates data between customer and business services objects in the delivery system 12 in the form that data is stored in the legacy applications. A different legacy app bridge component 84 may exist for each type of legacy application set, such as USCAT, AsiaCAT, LatinCAT, and EuroCAT. On entrance to a legacy application, the legacy app bridge component 84 obtains data from the session services set 130 and customer services set 110 and translates the data into the global data structures needed by the legacy application. On exit from a legacy application, the legacy app bridge component 84 takes modified data from the legacy structures and puts the data back to the customer services set 110 within the delivery system 12. The legacy app bridge component 84 translates legacy pages into the canonical page structures needed by the presentation manager component 52 and interfaces with the back door man component 101 to send messages to external service providers. The legacy app

bridge component 84 also interfaces with the logger for logging errors and transactions. During initialization of the legacy app bridge component 84, the rule broker component 121 and various rule authorities, primarily acquirer and issuer, may need to be interrogated to obtain data needed to populate static tables used by the legacy applications for processing rules. Depending upon the extent of migration, the legacy app bridge component 84 may have several different relationships between it and the navigation shell component 82. For instance, the navigation shell component 82 may provide the top level navigation across the new mini-app dialog component 83 as well as the individual legacy app bridge component 84. For some card types and issuers, the navigation shell component 82 may be faceless and all business functionality is provided by the legacy apps. In this alternative, top level navigation may be provided within the legacy applications. For CAT applications, one of a pool of CAT/TAFE runtimes will be assigned to a session at start-up. The legacy applications will be assigned a frame space within which the navigation shell component 82 "plays" its applications. Individual CAT level 3 functions will be individually registered and exposed. The navigation shell component 82 supports exposing CAT level 3 functions without the need to traverse the existing level 2 menu structure.

G. Transaction Services Set 90

The transaction services set 90 handles external service provider transactions needed to accomplish particular business functions. The components within the transaction services set 90 provide transaction coordination and external service provider message formatting. In some cases, more than one transaction executor component 91 may be associated with a given business function. Some examples of typical transaction executor components 91 are profile transaction component, scam transaction component, withdrawal component, deposit component, transfer component, transaction journal component, get payee list component, update payee list component, and make a payment component.

Each transaction executor component 91 performs a particular business function, such as cash withdrawal, by doing transactions with external service providers. The transaction executor component 91 validates properties of data obtained from mini-app dialog components 83 to determine whether the required information needed to do the transaction

exists. If the data is missing, the transaction executor component 91 immediately returns an error. The transaction executor component 91 collects additional information needed to do the transaction from other objects, such as customer ID component 111, acquirer component 114, issuer component 112, or account component 115. The transaction executor component 91 formats messages to be sent to external service providers and orchestrates complex transactions by sending messages to multiple service providers, serially or concurrently, as necessary. The transaction executor component 91 also parses response messages and returns information as properties of a transaction object and recovers from external service provider transaction failures. The transaction executor component 91 may also reverse transactions during a recovery. The transaction executor component 91 calls system logger component 72 to record an audit trail of transactions.

H. External Service Provider Interface Set 100

The external service provider interface services set 100 provides protocol support for interfacing with external service providers 22. The external service provider interface services set 100 includes the back door man component 101 and the external service provider interface component 102.

The back door man component 101 multiplexes messages from multiple transaction executor components 91 in several sessions to a single external service provider. The back door man component 101 provides message sequencing over all messages sent to a particular external service provider and also provides response routing back to the requesting transaction executor component 91. The back door man component 101 secures messages exchanged with an external service provider, such as with MAC or encryption. The back door man component 101 generates sequence numbers, adds external service provider envelope to outgoing messages, and sends outgoing messages to the external service provider interface manager. The back door man component 101 is responsible for retry of messages and checks sequencing of incoming messages. The back door man component 101 routes response messages to the proper transaction executor component 91 and routes incoming unsolicited messages to a registered or well-known system component. The back door man component 101 switches between alternate or back-up external service providers to provide

error recovery, load sharing, or alternate routing. The back door man component 101 can support multiple outstanding requests simultaneously. During operation, the back door man component 101 knows which of the alternate or back-up external service providers are active, the name/addresses of external service providers, server ID information, message sequence numbers, and message security context.

The external service provider interface manager component 102 provides protocol support for connecting to an external service provider 22. For example, the external service provider interface manager component 102 might provide X.25, 3270, or SNA protocol support. The external service provider interface manager component 102 provides protocol support for a specific type external service provider interface, if needed.

I. Customer Services Set 110

The customer services set 110 provides a category of services that includes all information specific to the customer who initiates a session. All information related to identifying the customer, the issuing business of the customer, the customer's profile, and all the customer's accounts are the component objects included within this category of services. The customer services set 110 includes a customer ID component 111, an issuer component 112, a customer relationship component 113, an acquirer component 114, and an account component 115.

The customer ID component 111 contains information and answers questions about a customer's identity and associated information. The customer ID component 111 supports query of customer ID and card information, supports update of customer ID and card information, and identifies card issuer. The customer ID component 111 knows the customer primary ID including the CIN, encrypted PIN/TPIN, and public key certificate. The customer ID component 111 also knows the status and profile action code indicating ID validity: valid, invalid, or unknown. The customer ID component 111 has card information, if a card was used, including the type of card, such as ATM, credit card, SmartCard, and tracks present and track data. The customer ID component 111 knows the tier of service a card supports, the advisory message text to be displayed, the primary relationship type code, and the deposit only flag. The customer ID component 111 has links to account list, an issuer list, and a

customer relationship list. The customer ID component 111 may also store the name of a customer, mail address of customer, E-mail address of customer, and phone numbers of the customer and provide this information to the customer or external service provider 22 so that this information does not have to be requested more than once.

The issuer component 112 represents the issuing business for the customer-ID information that was used to start a session. The issuer component 112 is the rule authority for all general, issuer related, non mini-app specific business rules. The issuer component 112 supports query of issuer information and supports answering questions about general issuer business rules. The issuer component 112 has information about the issuer of customer's identity, for instance, business code, financial institution identifier, and issuer type, such as bank card, credit card, or other third party card. The issuer component 112 knows the PIN length supported and the issuer country and ISO currency code for the issuer default currency. The issuer component 112 has a list of customer relationships for the issuer and a list of accounts for the issuer. The issuer component 112 also knows the products and services supported and the transaction and product limits. The issuer component 112 is informed of the issuer's presentation rules, such as data, format, and account number masking, and the issuer's local rules, such as collect call support, currency, and product names. The issuer component 112 also knows the issuer's servicer-ESP communication rules, for instance, profile message support, the languages supported, and the navigation schemes supported. The issuer component 112 knows when or how to authenticate customer, such as by local validation of public key certificate, immediate to issuer, background to issuer, or delayed to first transaction.

The customer relationship component 113 contains information and can answer questions about a customer's relationship. The information contained within the customer relationship component 113 includes the accounts and products owned by the customer, customer type, preferences and privileges. The customer relationship component 113 supports query of customer relationship information and supports update of customer relationship information. The customer relationship component 113 knows the owner of the customer relationship or issuer, the customer relationship ID, the customer relationship type, such as Citibank account or CitiGold, and the customer relationship nickname. The customer

relationship component 113 has a list of accounts/products associated with a customer, a list of account categories, and a list of accounts for the customer. The customer relationship component 113 also knows the customer's predefined transactions and has an account summary status. The customer relationship component 113 has the list of payees and the payee list status. The customer relationship component 113 knows the customer privileges or limitations, such as the number of quotes allowed for that customer. Some businesses, such as those in Mexico, Venezuela, or Brazil, can have multiple relationships per card. In the top level navigation, the customer may select one of them as the primary relationship to use for a session. The transfer application, however, can transfer between accounts in different relationships.

The acquirer component 114 contains information and answers about the acquirer. The acquirer component 114 represents the acquiring business for a session and is the rule authority for business rules that are acquirer related, but not mini-app specific. For rules that are acquirer related and mini-app specific, separate rule authorities may be registered as part as a dynamic installation of a mini-application. The acquirer component 114 supports query of acquirer information and processes certain specific rules associated with the acquirer. The acquirer component 114 knows information about acquiring business for a session, for instance a financial institution identifier and business code, and knows the country or region of acquirer.

The account component 115 contains information and can answer questions about a particular account. Each individual account preferably has only one account component 115 with the account details and rules varying for the particular individual account. The account component 115 supports query of account information and supports update of account information. The account component 115 knows the business owning the account, the category of the account, and the product type and subproduct type of the account. The account component 115 also knows the fund family code and fund code, the category code, the account name, account number, and account details, such as currency code, balances, and terms. The account component 115 has information on the functional privileges and limitations and also information on associated link accounts. The individual accounts may be customer owned or payee accounts that can be the target of a transfer or bill payment.

J. Business Services Set 120

The business services set 120 provides formal mechanisms for dealing with business rules, language support, and acquirer services. The business services set 120 includes a rule broker component 121 and a language man component 122.

The rule broker component 121 formalizes a mechanism for dealing with business rules that have conventionally been ad hoc. The rule broker component 121 is a central registry for all business questions. Other components within a session address named business questions to the rule broker component 121. The rule broker component 121 routes the question to the rule authority or authorities that have registered for a rule. By having a separable rule authority for each mini-app specific business rule, new rules can be added independently without affecting the rest of the delivery system 12. The rule broker component 121 supports the concept of overrides, which allow the dynamic registration of a new rule authority when changes to business rules are necessary. The rule broker component 121 may either answer questions directly or route questions to another component, such as an account component 115 or the issuer component 112. The rule broker component 121 is also responsible for interfacing into rules databases and knows what component will answer each question.

The language man component 122 provides the application with a facility to resolve the necessary text phrase needed in a particular context. The context includes the language selected by the customer and the type of device in use. The language man component 122 provides a repository of phrases which allows an application to be written in a language and device independent way. From the application point of view, all phrases are named. When an application needs to display a phrase, the application queries the language man component 122 for the correct text for this phrase name given a specified language choice and the current presentation device type, which has been provided by the presentation manager component 52. The language man component 122 can also extend this capability to the use of phrases with imbedded variables. Thus, the application may supply additional parameters to be inserted into the phrase at a required point. To resolve a request, the language man component 122 uses a phrase repository to look up the correct version of a particular phrase,

with the repository being segmented. A set of "global" phrases are usable by all applications and a mini-app dialog specific set of phrases is established. Thus, given the ID of a requesting mini-app dialog component 83, the repository specific to that mini-app dialog component 83 is searched first and then, if the phrase is not found, a global repository is searched. The phrase repository allows a degree of independence in the creation of mini-app dialog components 83. No coordinated update to the global repository is needed to release a new mini-app dialog component 83 and a mini-app dialog 83 can override the global phrase. The language man component 122 also provides APIs for the dynamic construction of phrases needed to deal with gender and plural issues encountered in some languages. The language man component 122 looks up a requested phrase in a phrase repository and returns the phrase based on the client ID, language ID, locale, phrase medium, phrase formulate, and device type and may be qualified by the device, as well, such as short form of the phrase for a small display on the device. The language man component 122 is backed by a set of development tools to create and maintain phrase repositories. These development tools provide for creation and deletion of phrase IDs, mechanisms to add, change, and delete phrase text in the repository, multi-lingual text entry, and specification of variable insertion points as well as graphic files or sound or video files.

K. Session Services Set 130

The session services set 130 includes a session controller component 131, a session component 132, and a delivery capabilities component 133. The session controller component 131 manages all the sessions in the delivery system 12. When a new customer contacts the delivery system 12, the session controller component 131 starts a session by instantiating a session bubble for the session. The session bubble, for instance session bubble S shown in Fig. 2, bounds a secure set of resources allocated to one and only one customer session. The session controller component 131 is aware of the type of customer remote device a start session request came from and the broad product type of service requested so that the appropriate type of session bubble can be instantiated. The session controller component 131 creates a session when a customer contacts the delivery system 12 by instantiating a new instance of the session object. The session controller component 131 maintains a registry of all active sessions with handles to the session objects. The session controller component 131 also terminates a session when a customer abnormally breaks the connection.

The session component 132 manages the resources associated with this session. The session component 132 brings up some initial session resources and is the registry for the brought up session components. The session component 132 also knows certain session context information as well as all assigned session resources and services. The session component 132 instantiates and initializes the following resources when a session is created and deletes them when the session is terminated: delivery capabilities component 133, rule broker component 121, front door man component 51, presentation manager component 52, acquirer component 114, language man component 122, and wome mat component 81. The session component 132 sends touch point attached notification to each of the components and supports registration of additional session components that need to be accessed globally by the session. The session component 132 recovers resources when a session abnormally terminates and logs significant session events, such as start or end of session and session errors. The session component 132 has session initiation information including the session ID and the start of session time. The session component 132 also has the handles for linking to many other session components and knows which navigation shell components 82 and

mini-app dialog components 83 are active. The session component 132 also knows the reason for the end of a session.

The delivery capabilities component 133 holds data and answers questions about the delivery capabilities of a remote device for a particular session. The information contained within the delivery capabilities component 133 is communicated either explicitly or implicitly in the start up message from the remote device causing the initiation of a session. The delivery capabilities component 133 is available for interrogation from other components within the delivery system 12. The delivery capabilities component 133 answers questions about the delivery capabilities of a remote device. For instance, for a web browser remote device, the delivery capabilities would include the HTML level, less preferably, FTP, picture formats, applet types, script types, and international fonts. The delivery capabilities component 133 is instantiated by the session controller component 131 with the initial capabilities based on access mode, for example, Internet, dial-in, or CAT.

II. Walk-Through Examples

A. Start of Banking Session

An example session will now be described with reference to Figs. 3A to 3C and Figs. 4A to 4C. At a step E1, a customer initiates a session. The customer may initiate a session in various ways depending upon the remote device used to communicate with the delivery system 12. For instance, the customer may use a screen phone 14, a CAT/CASST 16, a personal computer 18, or a PDA 20. The customer may also use a remote device 24 and an external service provider 22 to communicate with the delivery system 12. The customer, regardless of the particular remote device used, initiates the session through the touch point and display component 31 of the delivery system 12. At a step E2, a start banking message is sent from the touch point and display component 31 to the touch point interface component 41. At step E3, the touch point interface component 41 sends the start session message to the session controller component 131. At step E4, the session controller instantiates session component 132. At step E5, the session component 132 then instantiates the delivery capabilities component 133 and the session device manager component 63.

At step E6, the session component 132 instantiates the front door man component 51.

The session component 132 instantiates the presentation manager component 52 at step E6 and instantiates the presentation manager component 52 at step E7. At step E8, the session component 132 instantiates the rule broker component 121 and at step E9 instantiates the language man component 122. At step E10, the session component 132 instantiates the acquirer component 114 and at step E11 instantiates the wome mat component 81.

At step E12, the wome mat component 81 sends a logon to presentation manager component 52. The presentation manager component 52, at step E13, formats the screen based on device specific template and sends formatted information to the front door man component 51. At step E14, the front door man component 51 assigns a session cookie and sends a response via the touch point interface component 41 to the customer.

As reflected in steps E1 through E14, a customer can access the delivery system 12 with any type of remote device. In response, the delivery system 12 will create a session bubble specific for that customer. This session bubble will preferably have a session component 132, a delivery capabilities component 133, a session device manager component 63, a rule broker component 121, a wome mat component 81, a front door man component 51, as well as various other components dedicated for that particular session. Through the presentation manager component 52, front door man component 51, touch point interface component 41 and touch point and display component 31, the delivery system 12 can format messages to any type of remote device and can custom tailor this message according to the desires of a particular customer. The delivery system 12 is also capable of providing uniformity across the various remote devices so that the customer is presented with a consistent and familiar interface regardless of the remote device used.

B. Customer Authentication

An example of the process of authenticating a customer will now be described with reference to Figs. 5A to 5D and Figs. 6A to 6C. At step E21, a customer enters his or her CIN and PIN at the touch point and display component 31 which forwards the information to the touch point interface component 41. At step E22, the touch point interface component 41 forwards the message to the appropriate session bubble based on the session ID in the session cookie. At step E3, the front door man component 51 performs a security check on the

cookie and other parameters before forwarding the message to the presentation manager component 52. At step E24, the presentation manager component 52 routes the input to the dialog services set 80. For instance, the presentation manager component 52 may route the input based on mime type and URL to the appropriate dialog wome mat component 81.

At step E25, the wome mat component 81 asks the rule broker component 121 who is the issuer based on the CIN. The wome mat component 81, in turn, instantiates the customer ID component 111 at step E26 and instantiates the issuer component 112 at step E27. At step E28, the wome mat component 81 instantiates the profile transaction executor component 91 for authenticating the customer and then passes the CIN and encrypted PIN to the transaction executor component 91. At step E29, the transaction executor component 91 formats a reply message and sends the message to the host through the back door man component 101. At step E30, the back door man component 101 adds a universal message sequence and, at step E31, the external service provider interface component 102 provides protocol gateway to the external service provider 22.

At step E32, a response is returned to the back door man component 101 and the back door man component 101 routes the response to the appropriate transaction executor component 91. At step E33, the transaction executor component 91 extracts information from the external service provider message and gives this information to the wome mat component 81. At step E34, the transaction executor component 91 instantiates the customer relationship component 113 which, in turn, instantiates the account components 115 at step E35. At step E36, the wome mat component 91 instantiates the navigation shell component 82 which sends initial navigation choices to the customer at step E37. At step E38, the presentation manager component 52 formats style of screen display and sends a response to the customer via the front door man component 51, touch point interface component 41, and touch point and display unit 31.

C. Selection of Mini-App

The selection of a mini-app will now be described with reference to Figs. 7A and 7B and Figs. 8A and 8B. At step E41, the customer selects a mini-app with the touch point and display component 31 and the request is sent into the delivery system 12. At step E42, the

presentation manager component 52 demultiplexes the request based on mime-type and URL and sends the request to the navigation shell component 82. A step E43, the navigation shell component 82 instantiates the appropriate mini-app dialog component 83. At step E44, the mini-app dialog component 83 returns choices to the customer. At step E45, a back and forth dialog occurs between the customer and the mini-app dialog component 83 until all information is collected for a function. During this step, the mini-app dialog component 83 directs business rule questions to the rule broker component 121 for resolution during the dialog.

At step E46, after all information has been collected, the mini-app dialog component 83 instantiates the transaction executor component 91 for the selected function. At step E47, the transaction executor component 91 formats a message to the external service provider 22 and does the transaction with the external service provider 22. At step E48, the transaction executor component 91 extracts information from the host message and passes the information to the mini-app dialog component 83. As step E49, the mini-app dialog component 83 formulates content of response and sends the response to the presentation manager component 52 for formatting. At step E50, the presentation manager component 52 formats style and layout of response and sends the response to the customer via the front door man component 51, touch point interface component 41, and touch point and display component 31.

III. Rendering Model

To allow for both local delivery to the CAT 16 and to other remote devices, the basic rendering model is indirect. Preferably, none of the components within the dialog services set 80 draw directly to the screen but rather produce a stream of data, the app stream, that will ultimately be rendered by the touch point and display components 31. The app stream is preferably an HTML encoded stream of named objects or tokens with a named template or forms. The dialog services set 80 may then set the properties of these named objects within named templates. Although the dialog services set 80 may set any property of a named object, the delivery system 12 preferably separates content from style so that a specific mini-app can be leveraged and delivered across many delivery vehicles. In general, the mini-app

dialog component 83 will operate by setting the values of named properties of named objects and named templates, such as TemplateX.ObjectY.PropertyZ= Value.

The presentation manager component 52, using delivery vehicles specific named templates, is responsible for style and mapping to the encoding language of the target device. The presentation manager component 52 takes the app stream received from the mini-app dialog component 83 and, based on delivery vehicle specific templates, merges the data based on mapping rules and produces the final token stream that is sent to the touch point and display component 31. A one to one mapping exists between canonical templates that mini-app dialog components 83 reference and delivery vehicle specific templates that the presentation manager component 52 uses. Delivery vehicle specific templates include specific information on the layout, colors, and mapping of individual objects. A set of emerging standards from Microsoft and WC3 on advance style sheets, including style sheets that allow precise X, Y positioning of objects may be used as part of the templating mechanism.

Separation of content from style provides many benefits. For instance, separation allows the style and layout of a presentation to be defined and changed independent of the code in the mini-app dialog components 83. Also, separation allows a single mini-app dialog component 83 to deliver its functions to more than one target delivery vehicle through the abstraction of individual objects or tokens. The delivery system 12 allows and encourages the use of abstract objects in the app stream. For instance, the use of an abstract object like "choice" instead of a specific object like "button" allows the choice to manifest its in many ways on the target delivery vehicle. A choice could manifest its in one case as a CAT button, in another as a Windows style button, as an HTML anchor, or as an item in a scrolling list.

The delivery system 12 preferably supports ActiveX visual controls within delivery vehicle specific templates. The delivery system 12, however, is preferably expanded so as to map controls to alternative objects for presentation on delivery vehicles that do not support ActiveX controls. The delivery system 12 also encourages the grouping of logically related named objects into named groups. The grouping facilities allow the masking out of a group so that it will not be delivered to certain delivery vehicles based on the capabilities of that device or screen real estate.

The delivery vehicle specific templates define layout and style both for frame sets and within a frame. A frame is a well-known concept within Web browsers and is a rectangular portion of screen real estate, which may be bordered or borderless. A frame set defines the layout of frames within an overall screen window. The frame set defines the width and height of each frame and an initial link to the HTML page or program that will provide the content for that frame. The presentation manager component 52 manages the overall display. Based on templates, the presentation manager component 52 assigns a frame or frames to a navigation shell component 82. In turn, based on templates, the navigation shell component 82 assigns a frame to a mini-app dialog component 83. Within a frame, the layout of that frame is controlled by a delivery vehicle specific template. By assigning frames that bound the display space of specific mini-app dialog components 83, an independence between one mini-app dialog component 83 and another can be maintained and different navigation shell components 82 may be installed independently of the mini-app dialog component 83. The presentation manager component 52 will model the display space as a set of frames and, based on the delivery vehicle specific templates for non-framed devices, the presentation manager component 52 will merge information from many frames into a single frame for delivery to a remote device.

The canonical templates that mini-app dialog components 83 use are bounded by a frame. The mini-app dialog components 83 are responsible for setting the properties of the named objects within its canonical templates. One of these properties that the mini-app dialog component 83 is responsible for setting for "choice" objects is a link. A link is a standard universal resource locator (URL) that specifies the target object, such as the mini-app dialog component 83, and a list of parameters that will be returned if this choice is selected by the customer. The activation of links by the customer is one of the main ways of making choices and navigating through a mini-app dialog component 83. In addition to links, data entered by the customer in input fields, select lists, check boxes, radio buttons, as well as in other ways will be returned to the mini-app dialog component 83 in the app stream in the standard HTML encoding style of name-value pairs. The basic app stream interface can be produced with any programming language. For instance, any programming language that can produce a text stream can also produce an app stream. The programming language preferably

should be able to communicate via COM but otherwise has no restrictions. The app stream is a multi-channeled stream capable of supporting the basic text based app stream as well as other mime types.

Although the delivery system 12 encourages leveraging one mini-app dialog component 83 over a large range of delivery vehicles, the delivery system 12 does not preclude writing mini-app dialog components 83 targeted towards a specific delivery vehicle or class of delivery vehicles. The mechanism of passing the app stream between mini-app dialog components 83 and the presentation manager component 52 would remain the same. The mini-app dialog component 83 is still responsible for content and the presentation manager component 52 for style and layout. In this case, however, the range of visual object types or capabilities may only be available on a specific delivery vehicle and might not lend its to abstraction. For example, the inclusion of client-side scripting may only be available on certain devices or class of devices and may not be easily abstracted.

The delivery system 12 can easily support multi-media. HTML has well-known means for embedding and referencing a wide range of media types, for instance graphics, sounds, and movies. The delivery system 12 preferably uses standard HTML encoding techniques to incorporate this ever expanding set of media types into the delivery system 12 for use by remote devices. To support various error conditions and easy switching and restarting of mini-app dialog components 83, the presentation manager component 52 preferable caches the last page output for each frame that it manages.

IV. Mini-App Packaging

A fundamental advantage of the delivery system 12 is the independence of one mini-app dialog component 83 from another. The delivery system 12 provides a safe environment for the dynamic insertion and registration of new mini-apps with their navigation shell components 82. The delivery system 12 can introduce a new mini-app dialog component 83 so as to require a complete testing cycle only on the introduced application and not a regression test of the entire delivery system 12. The mini-app dialog components 83 are therefore preferably packaged as separate entities. For instance, a mini-app dialog component 83 may include an executable (.EXE) for the mini-app dialog component 83 including the

transaction executor component 91 as a DLL or object. The mini-app dialog component 83 also includes a rule file, including all new rule entries to be registered with the rule broker component 121. Also, when appropriate, the mini-app dialog component 83 includes a rules engine file per rule for any rules that can be interpreted by the general purpose rules engine and a rule database file per rule that supplies any needed data to support mini-app specific rule authorities. The mini-app dialog component 83 also includes a language file including all mini-app specific language phrases needed and, when appropriate, a template file containing all mini-app specific templates.

V. NetCAT

An example of a NetCAT server 200 is shown in Fig. 9. The NetCAT server 200 has the ability to present a traveling customer their "home screens." This ability is accomplished without the need to load CAT software for all regions on all CATS 16 around the world. The basic notion is to have at least one NetCAT server 200 for every region. On this NetCAT server 200, a region's CAT software will run and it will be capable of being "remotely projected" through any acquiring CAT 16 around the world, thus providing almost all of the customers home screens around the world. Differences from the customer's home screen will show up on the initial wome screen, until the customer's issuer is identified, and during certain transactions, notably cash withdrawal, where foreign exchange rates will have to be displayed and regulatory requirements of the acquiring country will have to be honored.

To start a NetCAT session, the traveling customer dips his or her card at "foreign" CAT 16 and a session bubble starts up normally at the CAT 16. When the wome mat component 81 determines that this customer is off-region, the wome mat component 81 makes a connection to the appropriate regional NetCAT server 200. The wome mat component 81 on the CAT 16 communicates with the session controller component 131 on the NetCAT server 200 to start up a session. The wome mat component 81 on the NetCAT server 200, after given card parameters upon start up, instantiates the customer ID component 111 and issuer components 112 on the NetCAT server 200. After NetCAT server 200 authenticates the customer, with its own external service provider, the NetCAT server 200 starts up a navigation shell component 82 on the NetCAT server 200. The CAT 16

exposes/copies certain of its components to the NetCAT server 200 for its use. The CAT 16, for instance, exposes the session component 132 , the acquirer component 114, the delivery capabilities component 133, the front door man component 51, the peripheral device manager component 62, the transaction executor components 91, and the wome mat component 81. The NetCAT server 200 uses these components for business rule inquiries, for delivery to CAT screen, for operation of the CAT peripherals, and for inquiry about the capabilities of the hosting CAT, such as fonts supported and pictographic printing. An example of a CAT 16 is shown in Fig. 10 with the exposed components marked with a black dot.

VI. Legacy Migration

The delivery system 12 supports an orderly migration of CAT functionality from implementation with AGS applications to implementation with service components on all platforms where AGS is used to delivery CAT look-and-feel functionality. An example of the interaction between CAT AGS applications and service components will be described with reference to Fig. 11. The AGS applications are executed within an instance of a TAFE process, the legacy run time AGS driver and associated functionality, and share a single persistent global data store. At the time a CAT application is invoked, session context is completely represented by the current state of the persistent global data and the content of the Exit message TAFE passes to the application. If this context can be instantiated by alternate means, then the business/customer functionality normally performed by the AGS level one and level two applications need not be performed before running a level three transaction application. At a high level, pretransaction session context is imported to the TAFE and a level three application is invoked with Exit message. After a return from level three application with Exit message, post transaction session context is exported from TAFE. For the case of a complete session performed in AGS, the interaction includes importing prelanguage selection session context to TAFE, invoking level two application with Exit message, returning from level two application with Exit message, and exporting post end-of-session context from TAFE.

The vehicle for import and export preferably is formatted messages that can be defined in an AGS data dictionary and received and sent by AGS applications. These

messages may be defined to be composed by the persistent global variables and tables that comprise the necessary context such that no data manipulation is required in AGS after receipt of the import message or prior to sending the export message. The delivery system 12 does not specify the handling of these message within TAFE or whether they are implemented as a single import message and a single export message per interaction. In general, the session services set 130 must capture and maintain sufficient session context information from which to derive the context representation required by the AGS application to be invoked. The required context will vary in detail by target AGS application set, such as USCAT, EuroCAT, AsiaCAT, LatinCAT and ICC. The required context will also vary with the application being invoked such as a level two or level three application. The legacy application bridge component 84, whether representing an AGS session or an individual AGS transaction application, preferable is capable of constructing and interpreting messages using the data name space appropriate to the target AGS application. The legacy app bridge component 84 embodies the knowledge of the other components it queries and in the specific properties it assesses in order to assemble the session context that it delivers to TAFE. Likewise, the knowledge of the components and properties that must be updated by the modified session context at the completion of an AGS processed transaction or session is also embodied by the legacy app bridge component 84.

The delivery system 12 is not limited to any particular manner for initiating a CAT AGS application. As an example, however, a pre-initialized TAFE AGS driver process is associated with the session bubble. Within the bubble, a faceless level one application waits on receipt of a start of session context message. The legacy app bridge component 84 for the customer selected transaction sends a start of session context message to the TAFE including track two data. The message sent to the TAFE preferably does not contain data from an element ID range specific to a card issuer. The level one application receives the message and updates session context and persistent global memory. Using the track two data, preinitialized static tables, and existing functionality, the level one application creates and sends the exit message to invoke the level two application appropriate to the card issuer. In this example, the level two application is a faceless, special purpose replacement for the original level two application. The level two application is specific to the element ID range

of the issuer and sends a request message for the remainder of the session context data. The request message is routed from the TAFE to the legacy app bridge component 84.

The legacy app bridge component 84 queries other service components in order to construct and return a response message containing the remainder of the session context, including data in the element ID range specific to the level two application that sent the request. The level two application receives the message and updates the session context and persistent global memory. Using the transaction type code, language code, and application state code received in the context data, together with existing functionality, the level two application creates and sends the exit message to invoke the level three application appropriate to the transaction type. The level three application processes the transaction and presents screens, sends and receives external service provider messages, device messages and logging messages, and updates session context in persistent global memory. Upon completion, the level three application sends an Exit message to return to the level two application. The level two application sends a message containing the updated post transaction session context which TAFE routes to the legacy app bridge component 84. The level two application also sends an Exit message to return to the level one application. The level one application waits in receipt of another start of session context message. The legacy app bridge 84 receives the post transaction session context and processes it causing the session context to be updated in the other appropriate service components. In this example, the level one and level two applications perform no customer or business functionality. The r of the level one and level two applications instead is preferably limited to receiving and returning context data and invoking the appropriate lower level application. The delivery system 12, however, can vary from that described above.

VII. Rule Broker

One advantage of the delivery system 12 is the separation of individually-installable business rules from the code embodied in the transactions specific components. Application components needing answers to rule questions asks the rule broker component 121 without knowing any details about how the rules are encoded and answered. The rule broker component 121 routes the question to the appropriate component which can supply an

answer. Components which supply rule answers may be installed independently of components which ask the rule questions. In addition, any data used by a rule "answerer" may be installed or replaced independently from components which use that data to determine answers to rule questions.

In general, a business rule is a statement of policy driven by business or regulatory needs, which defines context specific behavior of the applications. A business rule in the delivery system 12 may comprise any statement policy driven by business or regulatory needs, which defines context-specific behavior of an application. Business rules are discrete items which may be modified independently from other application components. Examples of business rules are choosing dispense amounts to display, maximum PIC retries, assignment of product types to summary categories, assignment of product types to product categories, and the number of account digits on print records. On average, fifty to one hundred business rules may exist per region with most of the rules being issuer rules and a fewer number of acquirer rules.

The rule broker component 121 is a single entity which components of the delivery system 12 may access to obtain answers to the business rule questions which affect application processing. The rule broker component 121 receives rule registration requests, registers rules in a rule registry, receives rule queries and routes them to the registered provider for that rule. The rule broker component 121 provides a mechanism for rule authorities to register themselves as answers for particular rule questions. When application components query the rule broker for a particular rule, the rule broker component 121 routes the query to the appropriate rule authority or to the rule engine. The rule broker component 121 its is not aware of the actual semantics of any of the rules. In the preferred embodiment, the rule broker component 121 is used by the mini-app dialog components 83, the transaction executor components 91, the presentation manager component 52, the navigation shell component 82, the wome mat component 81, and the legacy app bridge component 84. Although the delivery system 12 preferably includes the rule broker component 121, certain components within the delivery system 12 can be direct answerers of questions when appropriate.

The rule authority is a component which can answer rule questions. Components within the delivery system 12 act in the r of a rule authority component if they register themselves with the rule broker component 121 as the answerer of a named rule. For instance, the issuer component 112, the acquirer component 114, and the delivery capabilities component 133 may each be a rule authority. The rule authority components register rules with the rule broker component 121 and provide answers for the rules that they have registered. The rule authority components may access separately installable data to answer rule questions and this data may be separate from the rule registry information used by the rule broker component 121 and the rule engine.

The rule engine is a general rule interpreter. The rule engine can answer a rule query based on parameters passed in the query and some interpretable rule data in the rule database. Unlike rule authorities, the rule engine has no specific knowledge of rules or applications. The rule engine determines answers for rules and is used by the rule broker component 121 and calls the rule registry.

In operation, each rule registered with the rule broker will have a unique name which includes a version identifier. The name will be passed separately from other parameters in a rule query. All rule query parameters aside from the name will be passed in a s-defining way, for instance, a rule query may contain a name, type, and value for each parameter. Each rule registered with the rule broker will exist as an independent record in a rule registry. A rule in the rule registry will be defined as either data, such as an encoded string, which can be interpreted by a general rules engine or a rule authority which is registered to answer a rule. A component's registration of a rule may override a previous component's registration for that same rule. Each registered rule will define the expected type of parameters to be passed and rules can be dynamically added to the rule registry independently of all other rules.

The rule broker component 121 will route a rule query either to the general rules engine or to a sequence of rule authorities until an answer is obtained or no more authorities are available. The rule broker component 121 routes queries based only on the rule name and does not validate the parameter list. The rule authority or authorities are responsible for validating the parameters.

A preferred protocol exists between the rule broker and components querying the rule broker. Any component querying the rule broker must be prepared to handle the case of "no answer" gracefully. For example, a no answer may occur when no such rule is registered or when the component registered to answer the rule cannot answer. Also, the rule broker component 121 must return a specific "no answer" answer to a requester when no answer is available. Further, the rules engine and all rule authorities should dynamically check the parameter list and return the appropriate "no answer" if a discrepancy between the expected and received parameters exist.

A. Example 1, Dispense Amounts

A set of complex rules, spanning multiple configuration tables, is used in the AGS implementation when choosing what dispense amounts are displayed on selection buttons to a customer withdrawing cash at a CAT 16. The existing "withdraw cash" application is tightly coupled to the structure of these tables. The acquirer component 114 might register as a rule authority for the "WhatDispenseAmounts?" question. The input parameters for this question include the product type, which specifies the product being withdrawn from, and the currency. The output parameters include the result code and the variable length list of amounts. Some of the session data needed to answer the question, such as card type and level of service, is available from known session components and consequently is not passed as input. The acquirer component 114, in processing the request, may query whatever database contains specific rules for dispense amounts and ask the peripheral device manager component 62 to determine what denominations are available.

B. Example Two, Maximum PIC Retries?

As another example, a rule "MaxPICRetries?" to be processed by the rule engine is registered in the rule database. This rule has no input parameters and has output parameters of a result code and MaxPICRetries. As rule data, some interpretable data which indicates that a "business options" table should be searched for the MaxPICRetries value matching the session values of issuer and card type. All of the session data needed to answer the question, such as the issuer and card type, is available from known session components so no specific

input parameters are needed. The rule engine searches the specified table for a match on the session issuer and card type and returns the value of MaxPICRetries for that match.

VIII. Tools And Languages

The delivery system 12 is preferably language neutral. The applications can be written in any language which supports the object model used to specify the delivery system 12. Consequently, different components may be implemented in different languages and may migrate to a different language over time. As examples, VisualBasic, C++, and Java may be used in implementing the components of the delivery system 12.

The delivery system 12 is also not limited to any particular integrated development environment (IDE). The IDE, however, should have support for multi-user shared development and should have integration with a configuration management capability. The IDE should also support a tool "plug-in" capability to allow tools to be added which are unique to the delivery system 12. Some examples of these "plug-in" tools include configuration tools to allow for the maintenance of system configuration information and test tools including host and device emulators. Other tools include software distribution tools to standardize the method by which software upgrades are distributed, system management and logging tools, security protocols, and middle-ware for distributed object support in legacy system interfaces. Further tools include template development tools for both canonical and device specific templates, a rules database editor, services registry maintenance tools, and language man repository editor. The IDE preferably supports all of the selected targeted languages so as to minimize retraining and allows reuse of "plug-in" of tools across development languages. The operating system for the delivery system 12 is preferably Microsoft's Windows NT but may alternatively operate on other operating systems, such as a Macintosh or a UNIX operating system.

A component in the delivery system 12 may comprise any piece of hardware or software that is potentially independently replaceable with the software components being embodied as either executables (.EXEs) or dynamically loaded libraries (.DLLs). Components generally have well-defined interfaces. An application, in contrast, is a set of components that does a specific business function, such as cash withdrawal and may

comprise several components. Each application in the delivery system 12 preferably comprises one or more dialog components 83 for handling the user interface, one or more business rule components 121, and one or more transaction executor components 91 for handling the message interface with external service providers 22.

IX. System Management Services 300

An important aspect of an embodiment of the present invention involves remotely monitoring and managing hardware and software devices. Fig. 12 is a component object model which illustrates an example of key components and the relationship between the components for the remote system management aspect for an embodiment of the present invention. Referring to Fig. 12, the system management services 300 for an embodiment of the present invention include, for example, a management protocol agent 304, a command dispatch agent 306, and a status monitor agent 308. In addition, an embodiment of the present invention makes use, for example, of a system manager 302, a local management mini-app 310, a component registry 312, an event broker 314, a logger 316, a managed component 318, and an instrument 320.

The remote monitoring and managing aspect of an embodiment of the present invention makes use of instruments, such as instrument 320, that are intelligent software components. These instruments are sufficiently intelligent, for example, to report if a threshold is exceeded. The software components of these instruments reside, for example, in the applications in the ATM 16. The ATM 16 is used in this example because the infrastructure of which this aspect is a part can run on the ATM 16. However, the infrastructure can likewise run on other devices, such as home banking servers. There a number of aspects that form the infrastructure, including the instruments which form a part of the infrastructure. The instruments themselves are part of the infrastructure and are specific to whatever application the infrastructure is solving at a particular time. The concept of instruments is part of the infrastructure. At the same time, it is important that the instrument 320 is generic enough to provide the same interfaces, so that a system management agent, such as management protocol agent 304, command dispatch agent 306, or status monitor

agent 308, would favor a uniform approach to be informed about any changes and to inquire about the statuses of the instruments.

The infrastructure of which an embodiment of the present invention is a part is intended to become a uniform platform on which future applications will be built. An embodiment of the present invention is object oriented in its approach and consists of several building blocks, such as the instrument 320, which are usable and replaceable. An important feature of this aspect of an embodiment of the present invention is that regardless of the specifics of the application, the system management agent can handle events from ATMs, home banking servers, and the like, without modification. The instruments are logical components which are part of the infrastructure that are basically specific to an instrument, such as a cash replenishment instrument. For example, when a cassette that holds $20 bills for dispensing to customers reaches a certain level, it needs to be refilled soon, so it reports that it is low on cash. That instrument is responsible for that particular piece of data, and when the instrument reaches the threshold level, for example, it reports that it has reached the threshold to one of the system management agents.

In this aspect of an embodiment of the present invention, the instrument 320 can cause an alert to be generated at the agent to be delivered to a central location, where a management product receives the alert and, for example, displays it for an operations person. It can also be configurable to be sent to an even more centralized location which monitors the entire network, or it can be configured to be send the alert only to a local operator. The system for an embodiment of the present invention is quite generic and configurable and makes use, for example, of software components that can be arranged in different ways to solve the management problem of getting information about the status of a device or a system to a central location. What makes the components, such as the instruments, generic is that the interfaces are separate from the implementation, so that each instrument, regardless of its particular purpose, exposes a standard interface that allows the system management component to deal with that. The interface is well defined, so that the signature method for the interface is all that the system management agent needs to know. Based on its configuration, a decision is made as to what to do with the information that is received from the instruments.

An important feature of an embodiment of the present invention is that it provides a generic set of tools. A problem with current attempts to manage and monitor devices, such as ATMs, is that such attempts typically involve "hard coding" such that adding a new piece of information to be sent up from an ATM to a central site is quite difficult and requires a considerable amount of specific work. On the other hand, an embodiment of the present invention is part of an infrastructure which provides a homogeneous approach for a heterogeneous environment. Another problem with current attempts to manage and monitor devices is that they attach the issue of the communications to the central node, such that the design of local agents is very tightly coupled with the specific and typically proprietary protocols which are used for the communication. However, the system management aspect for an embodiment of the present invention emphasizes a component approach that decouples functions related to the sending of commands to the components receiving the events from the communications aspect and specific communications protocol. Therefore, if it is necessary to change the communication protocol which the agent uses to talk to the central office, such as system manager 302, it is only necessary to replug one of the components, and everything else works as before.

Another important feature of this aspect, relating to the particular architecture for an embodiment of the present invention, is the idea that the management protocol and all of the dealings with the management protocol are essentially separated out from the rest of the agent architecture, such that it can be replaced with something else. For example, if what amounts to a non-industry standard, proprietary protocol is in use, and if it is desired to communicate to an industry standard Simple Network Management Protocol (SNMP) management central server, it is only necessary to replace the one protocol agent with a different protocol agent to be able to communicate to a different management protocol. In an embodiment of the present invention, the switch from one protocol agent to another is transparent to other components, because all they need to deal with the particular unpluggable module is to use the defined interface and method signatures. What is behind those methods is inside the replaceable, repluggable module, which hides the communications decisions. This all falls into a componentized architecture which separates, in many ways, the implementation from the function.

System management services 300 for the remote monitoring and managing aspect of an embodiment of the present invention consists of three components, namely, the management protocol agent 304, the command dispatch agent 306, and the status monitor agent 308. The management protocol agent 304 interfaces with an external system management product, such as system manager 302, somewhere on the network. It translates a specific system management protocol to/from a format supported by the command dispatch agent 306 and the status monitor agent 308. The management protocol agent 304 translates an incoming management request into an inquiry or modify for the command dispatch agent 306, and translates a system management alarm from the status monitoring agent 308 into the remote system management protocol format.

The command dispatch agent 306 for an embodiment of the present invention sends management commands to managed components to control the managed component 318 or retrieve its status. It sends control/inquiry requests to a proper management component, such as a process controller or a session controller. The status monitor agent 308 monitors managed components and their instrumentation variables and events, determines if a local action is required or an external system management product, such as system manager 302, needs to be notified, and sends any important "alarms" to the external system management product 302 or generates a local action. The status monitor agent 308 registers for events with the event broker 314, registers with managed components for changes to their instruments, such as instrument 320, receives notification of changes in instrument values, sends "higher level alarms" to the management protocol agent 304, command dispatch agent 306 and/or event broker 314, and its state machine correlates and filters information.

Figs. 13, 14, and 15 are use case scenarios which depict examples of interactions between the key components and external system agents for an embodiment of the present invention. Referring to Figs.13, 14, and 15, references to CreateObject are invocations of ComponentFactory.CreateComponent ("ComponentName") unless otherwise noted in the description of the CreateInterface published by each specific component. Referring again to Figs. 13, 14, and 15, Fig. 13 shows a use case scenario which illustrates an example of system manager command processing for an embodiment of the present invention.

Referring to Fig. 13, at S1, a management component, such as session controller 322, sends a RegisterObject request (mys) to the component registry 312, and at S2, the component registry 312 sends an OK register response to the session controller 322. At S3, the sends a Create CounterNamed (ActiveSessions) request to the instrument 320, and at S4, the instrument 320 sends an OK – object reference response to the session controller 322. At S5, the system manager 302 sends an inquiry request to the management protocol agent 304, and at S6, the management protocol agent sends an inquire (SessionController) request to the command dispatch agent 306.

Referring further to Fig. 13, at S7, the command dispatch agent 306 sends an Obtain Component SessionController request to the component registry 312, and at S8, the component registry 312 sends an object reference response to the command dispatch agent 306. At S8, the command dispatch agent 306 sends an ExecuteCommand (Inquire) request to the session controller 322, and at S10, the session controller 322 sends a Collection of Instruments response to the command dispatch agent 306. At S11, the command dispatch agent 306 sends a Collection of Instruments response to the management protocol agent 304, and at S12, the management protocol agent 304 sends a response Status number active_sessions response to the system manager 302.

Fig. 14 shows a use case scenario which illustrates another example of system manager command processing for an embodiment of the present invention. At S15, the session controller 322 sends a RegisterObject (mys) request to the component registry 312, and at S16, the component registry 312 sends an OK register response to the session controller 322. At S17, the system manager 302 sends a Stop Command request to the management protocol agent 304, and at S18, the management protocol agent 304 sends an ExecuteCommand ("stop") to the command dispatch agent 306. At S19, the command dispatch agent 306 sends a Get Session Controller request to the component registry 312, and at S20, the component registry 312 sends a Session Controller reference response to the command dispatch agent 306. At S21, the command dispatch agent 306 sends an ExecuteCommand ("stop") request to the session controller 322 and an OK response to the management protocol agent 304. At S22, the management protocol agent sends an acknowledgment response to the system manager 302.

Fig. 15 shows a use case scenario which illustrates an example of status of cash status event processing for an embodiment of the present invention. Referring to Fig. 25, registration by the status monitoring agent 308 for events and counter observer is assumed to be performed at system startup. At S26, the counter (cass2 bills remaining) 324 sends a Report (cass1 very low) request to the cash dispenser 326, at S27, the cash dispenser 326 sends a Report (cass1 very low) to the status monitoring agent 308, and at S28, the status monitoring agent 308 sends an OK response to the cash dispenser 326. At S29, the cash dispenser 326 sends a Report (cass2 low) request to the status monitoring agent 308, and at S30, the status monitoring agent 308 sends an OK response to the cash dispenser 326.

Referring further to Fig. 15, at S30, the state machine of the status monitoring agent 308 correlates data and triggers a counter inquiry, and the status monitoring agent 308 sends a GetValue() request to the cash dispenser 326. At S31, the cash dispenser 326 sends a number of bills remaining in cass2 response to the status monitoring agent 308. At S32, the status monitoring agent 308 determines the true cash value and sends an Alarm (immediate replenishment needed) request to the management protocol agent 304. At S33, the management protocol agent 304 sends an immediate replenishment request to the system manager 302, and at S34, the management protocol agent 304 also sends an OK response to the status monitoring agent 308.

The management protocol agent 304 for an embodiment of the present invention is responsible for translating a remote system management protocol into a command, such as inquiry, stop, or start, supported by the command dispatch agent 306. The management protocol agent 304 also translates alarms into the remote system management protocol format for unsolicited alarms. A management protocol agent version exists for each external system management protocol supported, but only one instance of it is running on a given system.

In one aspect of an embodiment of the present invention, the management protocol agent 304 communicates with the remote system manager 302, through formatted messages. In an alternate aspect of an embodiment of the present invention, non-proprietary management protocol agents are used, such as an SNMP management protocol agent or Common Management Information Protocol (CMIP) management protocol agent. In that aspect, the agent consists at least in part of off-the-sh vendor code. The management protocol

agent 304 completely hides a particular protocol used to communicate with the remote system manager 302 from the other components of the system for an embodiment of the present invention.

The responsibilities of the management protocol agent 304 for an embodiment of the present invention include, for example, translating an incoming management request into a specific command, such as inquiry, stop, or start, for the command dispatch agent 306, translating an inquiry response from the managed component(s) into the remote system management protocol format, translating a system management alarm from the status monitoring agent 308 into the remote system management protocol format, and supporting secure access to a management server, such as authentication, privacy, and non-replication.

The management protocol agent 304 for an embodiment of the present invention publishes a number of standard interfaces, such as the standard IComponentIdentification interface. A management protocol agent external interface accepts system management requests from the external system manager 302. The exact form of management request is dependent on the system management protocol supported and can be distributed object-based or request-response message-based. An IProtocolTranslator interface of the management protocol agent 304 is exposed to internal system management components, such as the status monitor agent 308 and the command dispatch agent 306, for forwarding alarms and inquiry responses to the external system manager 302.

The management protocol agent 304 for an embodiment of the present invention makes use of a number of methods including for example, an Initialize method, an InternalEvent method, and a Finalize method. The Initialize method is called to start all internal initialization of the management protocol agent 304. Parameters for the Initialize method include ProcessControllerDispatchInterface which is a pointer to ProcessController's IDispatch interface. The return value is S_OK on success and a specific error code on failure. The completion status is reflected in the return value. The Initialize method raises an error if a parameter is missing. The subscribers list includes the status monitoring agent 308.

The InternalEvent method for an embodiment of the present invention is called for an internal event (alarm) to be forwarded to the external system manager 302. The management protocol agent 304 translates responses into the specific management protocol format.

Parameters for the InternalEvent method include EventName which is a name identifying an event. The parameters also include InstrumentValue which is the instrument value. The return value is S_OK on success and a specific error code on failure. The completion status is reflected in the return value. The InternalEvent method raises an error if a parameter is missing. The subscribers list includes the status monitoring agent 308.

The Finalize method for an embodiment of the present invention is called to stop all internal functionality of the management protocol agent 304. Parameters for the Finalize method include StopMode which is a value in seconds, the same as in StopProcess method of IprocessedLifecycle interface. The return value is $_OK on success or a specific error code on failure. The completion status is reflected in the return value. The Finalize method raises an error if a parameter is missing. The subscribers list includes the status monitoring agent 308.

The command dispatch agent 306 for an embodiment of the present invention is the central contact point for any management request originating from an external system manager, such as system manager 302, or local operator interface. A request is in the form of a command, such as inquiry, stop, or start, on a managed component 318. The command dispatch agent 306 accepts the request, obtains the managed component 318 from the managed component registry 312 and uses the IManagedComponent2 interface published by the managed component 318 to execute the requested command. For the inquire command, the command dispatch agent 306 can obtain, through the IManagedComponent2 interface, statuses of all instruments owned by a managed component 318.

Interfaces published by the command dispatch agent 306 include an IComponentIdentification interface and an ICommandDispatch interface. The ICommandDispatch interface provides services for the operator interface and the management protocol agent 304. The ICommandDispatch interface locates the managed component 318 and calls the ExecuteCommand( ) method of the IManagedComponent2 interface of the component.

The command dispatch agent 306 for an embodiment of the present invention utilizes an Initialize method, a ManagementRequest method and a Finalize method. The Initialize method is called to start all internal initialization of the command dispatch agent 306.

Parameters for the Initialize method include ProcessControllerDispatchInterface which is a pointer to the Idispatch interface of the ProcessController. The ManagementRequest method accepts a remote or local management command CommandName, such as inquiry, stop, or start, for a managed component ComponentName.

The Finalize method for an embodiment of the present invention is called to stop all internal functionality of the command dispatch agent 306. Parameters for the Finalize method include StopMode which is a value in seconds, the same as in StopProcess method of IProcessedLifecycle interface. The return value is S_OK on success or a specific error code on failure. The completion status is reflected in the return value. The Finalize method raises an error if a parameter is missing. The subscribers list includes the status monitoring agent 308.

Parameters for the ManagementRequest method for an embodiment of the present invention include ComponentName which is the name of a managed component 318, CommandName which is the name of a command, Params which is an optional parameter for the arguments of the command, and Response which is a pointer to a collection object containing the command response. For the "Inquire" command, the response is a pointer to collection of instruments name/value pairs. The return value is S_OK on success or a specific error code on failure. The completion status is reflected in the return value. The ManagementRequest method raises an error if a parameter is missing. The subscribers list includes the management protocol agent 304.

Other components needed by the command dispatch agent 306 for an embodiment of the present invention include an IObjectDirectory interface, an IManagedComponent2 interface, ICollection2, and IStatusMonitor. The command dispatch agent 306 uses the IObjectDirectory interface to get reference (interface pointer) to a specific managed component (findObject) or to get a pointer to a collection of all managed components contained in the ObjectDirectory object (ObjectNames). Each instrument 320 is owned by a managed component 318 which has the knowledge of all instruments it owns. The value of an instrument 320 is set and maintained by its owner, a managed component 318. The command dispatch agent 306 uses the IManagedComponent2 interface of the managed component 318 to inquire about instrument value.

The status monitoring agent 308 for an embodiment of the present invention is the delivery system component that reports unsolicited alarms to the system management server 302. The status monitoring agent 308 is responsible for correlating events generated by instruments, such as instrument 320, through managed components, such as managed component 318, which own these instruments or an event broker 314 and clock timers, in order to generate "higher level" system manager alarms and/or new event broker events. The status monitoring agent implementation is based on a rule driven state machine.

The responsibilities of the status monitoring agent 308 for an embodiment of the present invention include, for example, registering with managed components for changes of their instruments, registering for events from the event broker 314, maintaining a rule based state machine to correlate and filter information, and using events and timers as input to its state machine. New states may trigger actions, such as sending a "higher level alarm" to the management server 302 and/or event broker 314 and initiating a local action or inquiry through the command dispatch agent 306. The responsibilities of the status monitoring agent 308 also includes periodic polling of managed components.

Interfaces published by the status monitoring agent 308 for an embodiment of the present invention include an IComponentIdentification interface, an IProcessLifecycle interface, an IStatus interface, and an IEventNotification interface. The status monitoring agent 308 publishes the standard IComponentIdentification interface and the standard IProcessLifecycle interface. The IProcessLifecycle interface supports methods including an InitProcess method, a StartProcess method, a StopProcess method, a SuspendProcess method, a ResumeProcess method, and a ShutdownProcess method.

The status monitoring agent 308 publishes the IStatusMonitor interface to be used by other components to get the value of the overall system status which is maintained by the status monitoring agent 308 . The command dispatch agent 306 uses this interface to give the status monitoring agent 308 the interface pointer of the command dispatch agent 306 so that the status monitoring agent 308 informs the command dispatch agent 306 every time the overall system status changes. The status monitoring agent 308 publishes the IEventNotification interface to be used by managed components, including the event broker 314, as a sink interface to send unsolicited notifications of events and instrument changes. In

addition, the IEventNotification interface is used by the managed component registry 312, with which the status monitoring agent 308 registers for notification, to inform the status monitoring agent 308 about dynamic creation/deletion of the managed component 318.

The status monitoring agent 308 for an embodiment of the present invention makes use of an OnEventNotification method. The instrument 320 uses the OnEventNotification method for notification of its value update. Parameters for the OnEventNotification method include OnEventName which is the name of a event for which to register and Value which is the pointer to a collection of key/value pairs. The return value is S_OK on success or a specific error code on failure. The completion status is reflected in the return value.

The status monitoring agent 308 for an embodiment of the present invention makes use of an IProtocolTranslater interface, an IObjectDirectory interface, IEventRegistratioon interface, and an ICollection2 interface. The status monitoring agent 308 uses the InternalEvent() method of the IProtocolTranslator interface to forward higher level alarms to the remote system manager 302, through the management protocol agent 304. The status monitoring agent 308 uses a Component property of the IObjectDirectory interface to get a reference (interface pointer) to a collection of all managed components contained in the ObjectDirectory object. The status monitoring agent 308 uses the RegisterObserver() method of the IEventRegistration interface to register with a managed component 318 for notification of instrument changes for the instruments owned by the managed component 318.

The status monitoring agent 308 for an embodiment of the present invention is notified of changes in various instruments managed by the managed components. The status monitoring agent 308 processes this information based on its state machine instructions, passing it without changes in the simplest case, to send it to the management protocol agent 304. Examples of managed components which are observed by the status monitoring agent 308 include the session controller 322, a process controller, an encryption server, a back door manager, and a number of other components.

The status monitoring agent 308 for an embodiment of the present invention sends event notifications to the event broker 314. For example, the event name CashDispenser.StoppedOperating, describes and event in which the cash dispenser has ceased normal operation. A peripheral device manager notes that the cash dispenser has become

unavailable. A system management agent forwards the notification to the external system manager, the logger 316 logs the event, and the targets are notified asynchronously.

A number of foundation components collaborate to support the abstraction and testability of the remaining components of the delivery system architecture for an embodiment of the present invention. The component registry 312 hides the location of the implementation of components. A component factory uses the component registry 312 to locate component implementations in order to create instances of the components. The component factory also supplies each new component to a test manager, so that the test manager may wire its into those components that have been selected for testing or tracing.

The test manager for an embodiment of the present invention manages the testing and tracing of components in the system. The test manager collects information from the various components in the system by wiring its into the components during component creation. The components that have been wired for test report method entries and exits to the test manager during their operation. The configuration of which components are under test or trace can be driven by scripts or by an on-line test management user interface. The test manager can record the information reported by the components under test in a log, or the test manager can report the test results to the tester through the test management user interface.

Many components of the delivery system architecture for an embodiment of the present invention need to update counters or provide some means by which they may be monitored and controlled, especially components that need to be monitored and controlled by the system management facilities. Instruments allow interested components to observe changes in other components. Each instrument provides a point of contact or rendezvous between an instrument updater and its interested observers. Whenever an instrument updater changes the instrument value, the registered observers are notified of the change, giving them the opportunity to observe the changed instrument value. However, interested observers need not always register for change notifications. Some interested observers may access an instrument value as needed, instead of every time the instrument value changes. Whether an interested observer registers for change notifications depends on whether the observer needs to observe the instrument soon after it changes.

All instruments are created and maintained by an instrument manager for an embodiment of the present invention. Both instrument updaters and instrument observers obtain references to instruments from the instrument manager. The instruments are intended primarily, for example, for maintaining a persistent record of some aspect of the system state or some event counter, and dynamic observation of changes that occur in the system. A primary motivation behind instruments is the collection of management information, either for audit or system management. Components that do not need one or both of the primary aspects of instruments use other mechanisms in the system. For example, the event broker 314 is used to notify components of transient changes that require more information than a simple rendezvous provides.

A currently defined instrument set includes the various kinds of instruments, such as a counter instrument, a bounded counter instrument 348, a status instrument, and a control instrument. However, the instrument set is designed to be extensible. In addition, new kinds of more specialized instruments can be derived from the base instrument class and added to the standard instrument set. Each kind of instrument has a publisher that defines the name of the instrument and the values of the various instrument properties. The instrument manager creates and maintains instruments. The instrument maintains a value, registers and unregisters value observers, and notifies registered observers when value changes. The counter instrument increments or decrements a value. The bounded counter instrument 348 notifies registered observers when value crosses an upper or lower bound threshold. The status instrument publishes a list of the status values and names. The control instrument publishes a list of the control values and names.

Fig. 16 is a component object model which illustrates an example of a component factory and its collaborating components and the relationships between them for an embodiment of the present invention. Referring to Fig. 16, in addition to a component factory 328, an embodiment of the present invention makes use of a test management GUI 330, the test manager 332, a client component 334, the component registry 312, a created component 336, and a testable component 338. Fig.17 is a component object model which illustrates an example of instruments for an embodiment of the present invention. Referring to Fig. 17, an embodiment of the present invention makes use of the instrument 320, an

instrument manager 340, a status instrument 342, a control instrument 344, a counter instrument 346, a bounded counter instrument 348, an updater 350, and an observer 352.

Fig. 18 is a component object model which illustrates an example of a managed component for an embodiment of the present invention. Referring to Fig. 18, in addition to the managed component 318, an embodiment of the present invention makes use of the instrument manager 340, the control instrument 344, the status instrument 342, the command dispatch agent 306, and the status monitor agent 308. Fig. 19 is a component object model which illustrates an example of a monitored component for an embodiment of the present invention. Referring to Fig. 19, in addition to the monitored component 354, an embodiment of the present invention makes use of the instrument manager 340, the counter instrument 346, the status instrument 342, and the status monitor agent 308.

Fig. 20 shows a use case scenario which illustrates an example of the component factory startup for an embodiment of the present invention. Referring to Fig. 20, at S40, the process controller 356 sends a CreateObject("ComponentFactory") request to the component factory 328, at S41, the component factory 328 sends a CreateObject("ComponentRegistry") request to the component registry 312, and at S42, the component registry 312 sends a ComponentRegistry response to the component factory 328. At S43, the component factory 328 sends a CreateObject("TestManager") request to the test manager 332, and at S44, the test manager 332 sends a TestManager response to the component factory 328. At S45, the component factory 328 sends a ComponentFactory response to the process controller 356. At S46, the process controller 356 sends an InitComponent(Me) request to the component factory 328, and at S47, the component factory 328 sends a response to the process controller. At S48, the process controller 356 sends a StartComponent( ) request to the component factory 328, and at S49, the component factory 328 sends a True response to the process controller 365.

Fig. 21 shows a use case scenario which illustrates an example of create component process for an embodiment of the present invention. Referring to Fig. 21, at S55, the client component 334 sends a CreateComponent(ComponentName) request to the component factory 328, at S56, the component factory 328 sends a ClassNameComponent(ComponentName) request to the component registry 312, and at S57,

the component registry 312 sends a String response to the component factory 328. At S58, the component factory 328 sends a CreateObject(ClassName) request to the created component 336, and S59, the created component 336 sends a CreatedComponent response to the component factory 328. At S60, the component factory 328 sends an IsTestable(ComponentName) request to the component registry 312, and at S61, the component registry 312 sends a response to the component factory 328.

Referring further to Fig. 21, at S62, the new component is wired for test only if it is testable, and the component factory 328 sends a WireComponentForTest(Created Component) request to the test manager 332. At S63, the test manager 332 sends a SetManager request to the created component 336. At S64, the new component is wired for test only if it is under test, and the created component sends a response to the test manager 332. At S65, the test manager 332 sends a response to the component factory 328, and at S66, the component factory 328 sends a CreatedComponent response to the client component 334.

Fig. 22 shows a use case scenario which illustrates an example of an instrument rendezvous for an embodiment of the present invention. Referring to Fig. 22, at S70, the updater 350 sends a GetCounterNamed request to the instrument manager 340, and at S71, the instrument manager 340 provides access to the counter which it created internally and sends a CounterInstrument response to the updater 350. At S72, the observer 352 sends a GetCounterNamed request to the instrument manager 340, and at S73, the instrument manager 340 provides access to the counter which it created internally and sends a CounterInstrument response to the observer 352. At S74, the observer 352 sends a RegisterObserver request to the counter instrument 346, and at S75, the counter instrument 346 sends a response to the observer 352.

Referring further to Fig. 22, at S76, the updater 350 sends a Let Value = 5 request to the counter instrument 346, and at S77, the counter instrument 346 sends an ObserverUpdated request to the observer 352. At S78, the observer 352 sends a Value request to the counter instrument 346, and at S79 sends an Integer (5) response to the observer 352, and the observer 352 will likely do something interesting as a result of the

value change. At S80, the observer 352 sends a response to the counter instrument 346, and at S81, the counter instrument 346 sends a response to the updater 350.

The component factory 328 for an embodiment of the present invention creates new components for clients. Some of the components created are testable. The component factory 328 supplies those new components that are testable to the test manager 332 so that it may wire its into components that are under test. The component factory 328 publishes a component factory interface, created by the process controller 356, which establishes a private internal reference to the component registry 312. The component factory interface creates new instances of components and supplies new instances of testable components to the test manager 332 for testing and/or tracing. The component factory 328 makes use of a CreateComponent method which creates a new component for the client. The parameter for the CreateComponent method is ComponentName which is a string that contains the well-known name of a component. The return value responds with a new instance of the named component (as object). An error is raised if the ComponentName does not exist in the component registry 312, or if the named component cannot be created.

The component factory 328 for an embodiment of the present invention makes use of subscribed interfaces, including a good neighbor interface, a test support interface, and a component registry interface. The test support interface includes a Snapshot method and a SupportedTestResults method. The Snapshot method responds with a string that contains a snapshot of the state of the component. The parameter for the Snapshot method is InBrief which is a Boan that indicates whether the component should generate a brief or full snapshot of its internal state. The return value responds with a string that contains a snapshot of the component's internal state. The test manager 332 can use this method to obtain a snapshot of a testable component. The test manager 332 can report the snapshot to the user or log the snapshot in a trace or debug log.

The SupportedTestResults method for an embodiment of the present invention responds with the name - value pairs that identify the test results supported by the testable method named MethodName. The parameter for the SupportedTestResults method is MethodName which is a string that contains the name of a testable method, i.e., the name of one of the methods published by the component in its TestableMethodNames property. The

return value responds with a collection of name - value pairs. Each name - value pair identifies one of the test results supported by the testable method named MethodName. An error is raised if the MethodName is not one of the testable methods listed in TestableMethodNames.

The test manager 332 for an embodiment of the present invention typically uses the SupportedTestResults method to build a test result map for each of the components under test. The test manager 332 uses the test result maps to supply test results during component testing. Each testable component publishes the test results it supports for each of its testable method. Thus, each component establishes a contract with the test manager 332 regarding the kinds of built-in tests it supports.

The component registry 312 for an embodiment of the present invention maps well-known component names to fully-qualified class names. The component registry 312 also knows which components are testable. The component registry 312 publishes a component registry interface, created by the component factory 328, which reads the definitions of component name maps from a configuration file. The component factory 328 makes use of a ClassNameForComponent method and an IsTestable method. The ClassNameForComponent method responds with the fully-qualified class name that corresponds to the supplied ComponentName. The parameter for the ClassNameForComponent method is ComponentName which is a string that contains the name of a well-known (i.e., registered) component. The return value is a string that contains a fully-qualified class name. An error is raised if the ComponentName is not the name of a registered component. The component factory 328 uses this method to obtain the names of classes.

The IsTestable method for an embodiment of the present invention responds with whether the component named ComponentName is a testable component. The parameter for the IsTestable method is ComponentName which is a string that contains the name of a well-known (i.e., registered) component. The return value indicates whether the named component is testable. An error is raised if the ComponentName is not the name of a registered component. The component factory 328 uses this method to decide whether to supply a new component instance to the test manager 332 for testing or tracing.

The test manager 332 for an embodiment of the present invention knows which components are under test (or trace) and wires its into new components supplied by the component factory 328 for test (or trace). The test manager 332 publishes the test manager interface, created by the component factory 328, which initializes its internal tables that identify which components are under trace and which are under test. The test manager 332 makes use of various methods including a WireComponentForTest method, a TraceComponentNamed method, and a TestComponentNamed method.

The WireComponentForTest method for an embodiment of the present invention wires its into the supplied component if the component is under trace or test. A parameter for the WireComponentForTest method is TestableComponent which is an object that implements the test support interface. An error is raised if the TestableComponent does not implement the test support interface. 'The component factory 328 uses this method to supply a new component instance to the test manager 332 for testing or tracing. The TraceComponentNamed method updates the ComponentsUnderTrace, i.e., the component name is added or removed. The parameters for the WireComponentForTest method include ComponentName, which is a string that contains the name of a well-known (i.e., registered) component, and TraceOn, which is a Boan that indicates whether to turn tracing on (TRUE) or off (FALSE). The test management GUI 330 uses this method to turn tracing on and off for a specific component.

The TestComponentNamed method for an embodiment of the present invention updates the Components UnderTest, i.e., the component name is added or removed. Parameters of the TestComponentNamed method include ComponentName, which is a string that contains the name of a well-known (i.e., registered) component, and TestOn, which is a Boan that indicates whether to turn testing on (TRUE) or off (FALSE). The test management GUI 330 uses this method to turn testing on and off for a specific component.

In an embodiment of the present invention, testable components make use of several methods including an EnterMethod, a LogLabeledValue method, and an ExitMethod. Testable components use these methods in the test manager 332 to report testable method entry and exit, and to log values that are changed during the method. With respect to EnterMethod, the test manager 332 records the entry of the named method in the

TestableComponent and responds with a test result value. Parameters for EnterMethod include TestableComponent, which is an object that is one of the components under test (or trace), and MethodName, which is a string that contains the name of a testable method in the component under test (or trace). The return value is one of the supported test results of the testable method. Testable components use EnterMethod to report the entry of a testable method and to obtain instructions from the test manager 332 regarding what the method should do when the component is under test.

With regard to the LogLabeledValue method for an embodiment of the present invention, the test manager 332 appends an entry containing a labeled value to the trace test log. Parameters for the LogLabeledValue method include TestableComponent, which is an object which is one of the components under test (or trace), MethodName, which is a string that contains the name of a testable method in the component tinder test (or trace), Label, which is a string that contains the value label, such as "SomeValue =", and Value which is an integer value. Testable components use LogLabeledValue to log values in the test trace log generated by the
test manager 332.

With respect to ExitMethod, the test manager 332 records the exit of the named method in TestableComponent. Parameters or ExitMethod include TestableComponent, which is an object that is one of the components under test (or trace), MethodName, which is a string that contains the name of a testable method in the component tinder test (or trace), and MethodResult, which is a variant that contains the result of the method. Testable components use ExitMethod to report the exit and result of a testable method to the test manager 332.

In an embodiment of the present invention, the instrument 320 provides an abstract base for all derived instruments. No instances of an instrument 320 are ever created. Only instances of components that implement the IInstrument interface are ever created. The instrument publishes the Instrument interface and an IComponentIdentification interface. The instrument 320 makes use of an IEventNotification interface, which must be implemented by an EventNotification object to receive event notifications from an instrument, such as a counter instrument 346, a status instrument 342, or a bounded counter instrument 348.

Instrument 320 also makes use of an OnEventNotification method. Each instrument 320 uses this method to notify registered observers that the instrument value changed. The parameter for the OnEventNotification method is UpdatedInstrument which is an object that refers to the instrument whose value changed. This method gives the observer 352 the opportunity to note the change to the instrument value.

In an embodiment of the present invention, in addition to supporting the instrument value, a counter instrument 346 supports the ability to increment (and decrement) the instrument value. The change of the value and the observer notifications are accomplished as a single atomic operation. In other words, the value cannot be updated again until all the registered observers have been notified of the change. The name and meaning of a counter instrument 346 are defined by the updater 350. Thus, the component that plays the r of the updater 350 is considered the publisher of a counter instrument 346. A managed component 318 creates instances of the counter instrument 346. When created, the instance is initialized with the information it needs to locate its value.

The counter instrument 346 for an embodiment of the present invention makes use of an IInstrument interface which supports the access and update of the instrument value. Whenever a component updates the value (e.g., using Let in Visual Basic), all registered observers are notified of the change. The counter instrument 346 makes use of a ChangeValueBy method which changes the value of the instrument and notifies the EventNotification object of the change. The parameter for the ChangeValueBy method is Difference which is a long integer that is the amount by which to change the instrument value. The ChangeValueBy method notifies the EventNotification object after updating the instrument value. The EventNotification object then relays the notification to all its registered observers.

In an embodiment of the present invention, in addition to supporting the instrument value and the ability to increment and decrement the instrument value, a bounded counter instrument 348 supports the specification of a value range that modifies the change notification mechanism. Instead of notifying registered observers whenever the instrument value changes, the registered observers are notified when the instrument value crosses either the LowerBound or the UpperBound. If the AutoReset option property is TRUE, the

instrument value is automatically set to the InitialValue after all the registered observers have been notified. The name and meaning of a bounded counter instrument 348 are defined by the updater 350. Thus, the component that plays the r of the updater 350 is considered the publisher of a bounded counter instrument 348. As such, the updater 350 needs to establish the values of the properties of a bounded counter instrument 348. A managed component 318 creates instances of the bounded counter instrument 348 for an embodiment of the present invention. When created, the instance is initialized with the information it needs to locate its value. The bounded counter instrument interface supports the access and update of the instrument value. The ICounterInstrument interface supports the updating of the instrument value using the ChangeValueBy method.

In an embodiment of the present invention, in addition to supporting the instrument value, a status instrument 342 supports the ability to define a value StatusMap. The StatusMap contains a collection of name-value pairs that describe the values taken on by the instrument value. The name and meaning of a status instrument 342 are defined by the updater 350. Thus, the component that plays the r of the updater 350 is considered the publisher of a status instrument 342. As a result, the updater 350 must also establish the StatusMap for a status instrument 342. Status instruments can be used to report strings as well as values through the StatusMap. Because the updater 350 is also the instrument publisher, the updater 350 can update the StatusMap to contain new name-value pairs as needed. Thus, the semantics of a status instrument are flexible.

The instrument manager 340 for an embodiment of the present invention creates instances of the status instrument privately instead of using CreateObject. When created, the instance is initialized with the information it needs to locate its value. The instrument interface supports the access and update of the instrument value. Whenever a component updates the value (e.g., using Let in Visual Basic), all registered observers are notified of the change.

X. Conclusion

The delivery system 12 advantageously provides a common application base for customer activated applications for all remote devices. Thus, a financial institution need not

have a first delivery system for its ATMs, a second delivery system for its staff tellers, a third delivery system for personal computers or PDAs, and a fourth delivery system for external service providers. Instead, home banking devices such as a personal computer 18, a smart phone 14, an Internet browser remote device 24, and a PDA 20 may all access the books of a financial institution through the delivery system 12. In addition, the delivery system 12 may provide financial services to its customers through its CAT/CASST 16 and to its employees through branch and CSR staff platforms 26.

The delivery system 12 supports convergence to a base set of reusable global application components. These components may be reassembled in different combinations and organizations to form application suites or they may be customized for the environment in which they are used. Furthermore, the global application component may be complemented by components from a local business.

The delivery system 12 provides state of the art user interfaces. The interfaces provided by the delivery system 12 support integration of standard multi-media elements, such as pictures, video, and audio. The interfaces also support customizations needed for specific devices, languages, countries, and other local business needs. The interfaces further support multiple co-existing application navigation paradigms and also support the user working in multiple application components at a single time.

The delivery system 12 substantially improves development and maintenance cycle time. The delivery system 12 uses prefabricated components and templates instead of "from scratch" development. The delivery system 12 may embrace widely accepted industry standards for component interfaces so that off the sh "plumbing" may connect components and enable plugging-in third party components. The delivery system 12 supports "plug and play" application components that can automatically configure themselves for the environment and automatically insert themselves into top level navigation menus. The delivery system 12 supports high productivity prototyping/development tools for top level navigation definition and user interface design, making use of predefined look-and-feel standards. The delivery system 12 separates different parts of an application so that changes in one part do not affect other parts.

The delivery system 12 provides a smooth gradual migration from legacy applications into a new architecture. The delivery system 12 supports the harmonious coexistence of software built under the delivery system 12 along with existing legacy AGS applications. As a result, financial institutions do not need to introduce a totally new system but rather may leverage their existing legacy AGS applications while taking advantage of the delivery system 12.

In addition, the system monitoring and management aspect of the present invention provides a method and system for remotely monitoring hardware and software devices, which makes use of an agent set that provides a communication mechanism which enables the external system management product 302 to query a network of ATM's or home banking servers for their status. The status of each device is monitored using instrumentation software resident on the devices. The system is componentized and thus configurable because of a separation of implementation and function. The system makes use, for example, of the management protocol agent 304, which interfaces with the external system management product 302, and the command dispatch agent 306, which accepts requests from the management protocol agent 304, executes the requested command, and inquires about instrument values and statuses of all instruments owned by a managed component 318.

The system monitoring and management aspect also makes use of the status monitor agent 308, which monitors the managed components and their instrumentation variables and events, registers with managed components, for example, for changes of their instruments, maintains a state machine which may trigger various actions, periodically polls managed components, publishes a status monitor interface to be used, for example, by other components to get the value of the overall system status, publishes an event notification interface to be used, for example, by managed components to send unsolicited notifications, uses a component property of the object directory interface, for example, to get reference to a collection of all managed components contained in the object directory object, uses a register observer method of an event registration interface, for example, to register with a particular managed component, and sends alarms and inquiry responses to the external system management product via the management protocol agent 304.